AutoLISP 標準教科書
2027 版

AUTODESK
Authorised Developer

中森 隆道 (Nakamori Takamichi)

本書のご使用について

❶ 著作権に関して

本書は米国 Autodesk 社が提供する AutoLISP と DCL(ダイアログ コントロール言語) の解説書ですが、この AutoLISP と DCL は米国 Autodesk 社の登録商標です。
また、本書の内容及び掲載したプログラムの著作権は著者にあります。
個人で使用される以外のご使用に関しましては、著作権法の範囲内でのご使用をお願いいたします。

❷ 免責に関して

本書の内容については正確な記述に努めましたが、著者および出版社は本書の内容とサンプルプログラムに関しましては何ら保証するものでは無く、また運用した結果について責任を負いません。

AutoCAD のヘルプからの引用文には、[出典：AutoCAD2026 ヘルプ] と明記しておりますが、内容を損なわない範囲で筆者の言葉で書き換えた箇所もございます。

❸ お問い合わせに関して

本書の内容に関するお問い合わせは、以下のメールアドレスでお受けいたします。
autolisp@ellipse.ne.jp

❹ 内容の訂正に関して

本書の内容の訂正やプログラムの修正は、以下の Web に記載いたします。
https://www.ellipse.ne.jp/autolisp.html

❺ 掲載したプログラムの取得方法に関して

本書に掲載したプログラムは、以下の Web からダウンロードできます。
https://www.ellipse.ne.jp/autolisp.html

本書のプログラムは、AutoCAD2026 で作成しました。
またプログラムの検証は、AutoCAD2024 と AutoCAD2025 で行いました。

目　次

序　章

第 1 部　AutoLISP

第 1 章　AutoLISP の基本

２章　AutoLISP の関数

第３章　AutoCAD と対話する

第４章　オブジェクトを操作する

第２部　DCL(ダイアログ コントロール言語)

第１章　ダイアログとタイルの構造

第２章　タイルの機能とコールバック関数

第３章　タイルをコントロールする

第４章　ダイアログをコントロールする

INDEX

第 1 部　AutoLISP

第 1 章　AutoLISP の基本

AutoCAD のカスタマイズ用言語である AutoLISP は
他の開発言語とどのように違っているのでしょうか。

この章では以下のことを学びます。

第 1 節 AutoLISP & DCL

1 AutoLISP とは

1 AutoLISP の歴史

AutoLISP は LISP をベースとしたプログラミング言語で、AutoCAD の機能拡張の作成と利用に使用できます。
この AutoLISP は 1986 年から AutoCAD で使えるようになりましたが、1997 年の AutoCAD R14 から、基本関数の拡張や ActiveX への対応、リアクターといった機能がアドオンとして追加されました。これらの追加された関数には vl- 、vlax-、vla-、vlr- と接頭辞が付きます。
Autodesk はこれらの言語の拡張と開発環境も含めて Visual LISP と呼び、1999 年の AutoCAD 2000 からは標準で実装されました。

2 AutoLISP の特徴

①プログラムは何らかの値を返す関数の集合体です。
②アルゴリズムとデータは記述上リストと言われる同じ構造をしています。
③関数をデータのように扱えて、関数をプログラム実行中に生成することができます。
④リストと高階関数が使えます。

3 VisualLISP の特徴

Visual LISP は AutoLISP での開発を支援する統合開発環境（IDE ＝ Integrated Development Environment）です。
このため Visual LISP はプログラミング言語ではなく、開発環境を指しているとも言えます。

AutoLISP のプログラム自体は、一般のテキストエディタでも書くことができますが、Visual LISP 環境を使用すると次のような利点があります。
①プログラム開発中、AutoCAD にプログラムを簡単にロードしてテストすることができます。
②プログラムのステップ実行や変数値の監視などデバックに欠かせない機能を利用できます。
③テキストエディタには、関数の検索機能や、コードを見やすくするための色分け、カッコの対応などプログラマを支援する機能が含まれています。
④ AutoCAD の図面データベースの内容を表示させることができます。
⑤プロジェクト機能を使って、複数のファイルにまたがるプログラムの管理が容易に行えます。
⑥ VLX アプリケーションといった扱いが容易なプログラムにパッケージ化することができます。

VisualLISP ファイルの拡張子		
拡張子	形式	説 明
FAS	バイナリー	コンパイルされたプログラムファイル。
VLX	バイナリー	コンパイルされたアプリケーションファイル。
PRJ	テキスト	複数のファイルにまたがるプログラムを管理するプロジェクトファイル。
PRV	テキスト	LISP や DCL などをまとめて、一つのパッケージを作成します。

2 Dialog Control Language (DCL) とは

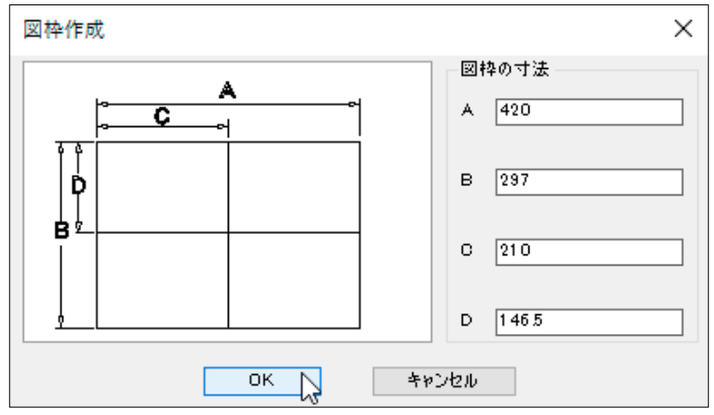
1 Dialog Control Language (DCL) の特徴

ダイアログコントロール言語または略して DCL は、AutoLISP および Visual LISP の作成者がルーチンに統合できるダイアログボックスを作成できるようにするマークアップ言語です。
つまり、AutoLISP ルーチンは、使い慣れた論理的なインターフェイスを介して、ユーザーからさまざまな入力データを収集できます。
DCL がない場合、AutoLISP はコマンドラインからのみユーザー入力を取得できます。

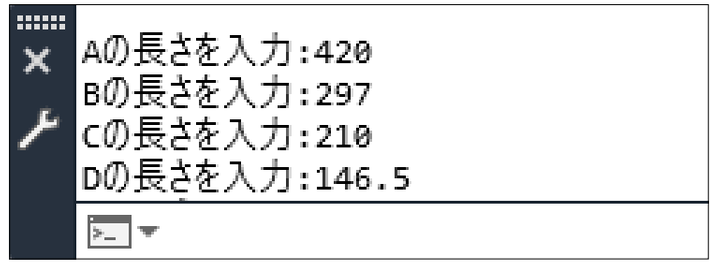
2 DCL を併用した AutoLISP のプログラム

下図のダイアログは、第 2 部 3 章 4 節で解説しているダイアログ ボックスです。
このダイアログ ボックスに A、B、C、D の数値を入力すると、左の図のような図枠が作成されます。

第 2 部 3 章 4 節



もしダイアログ ボックスを使用しない場合は、コマンドラインに以下のようなメッセージが表示されます。
コマンドラインに文字だけを表示するよりは、ダイアログ ボックスを使うほうが、より視覚的に分かりやすく、従って間違いも少なくなります。

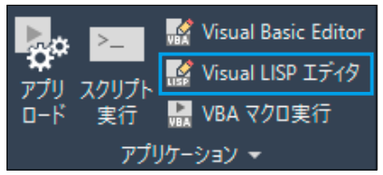


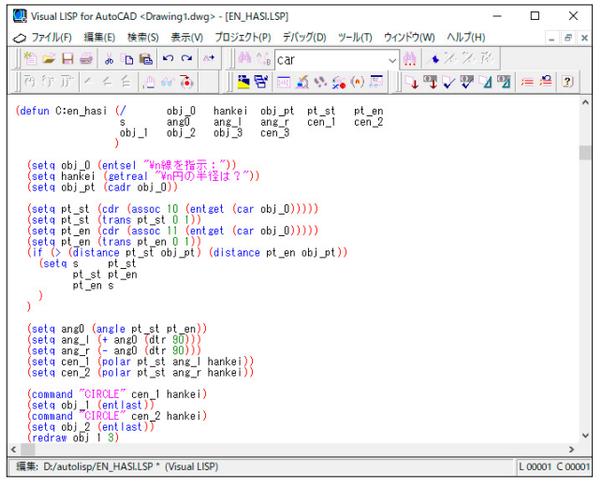
3 VisualLISP エディタ

AutoLISP を作成するために、2 つのエディタが提供されています。
従来からの [VisualLISP エディタ] と 2021 版から使用できる [Visual Studio Code] です。

1 VisualLISP エディタを利用する

①[管理] タブ -> [アプリケーション] パネル -> [Visual LISP エディタ] から開きます。



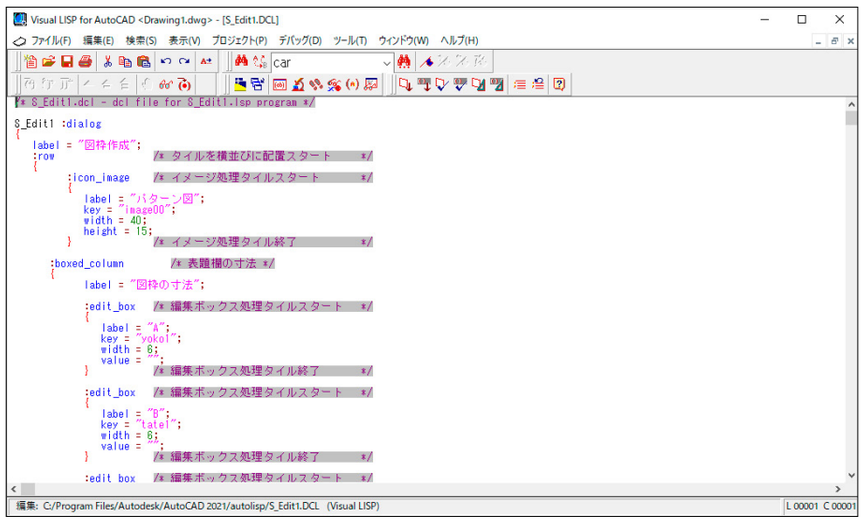


AutoCAD 2020 以前 のエディタ

日本語が正しく表示されないときは、プルダウンメニューの [ツール] → [ウィンドウの属性] → [フォント] から、日本語を選択します。

括弧や変数、文字が色分けして表示されます。
編集可能なファイルは、LISP、DCL、SQL、C/C++、SLD ファイルです。

② AutoLISP だけでなく、DCL の作成も可能です。

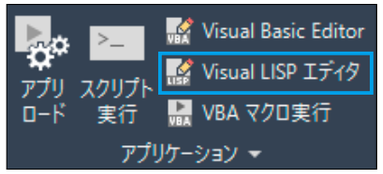


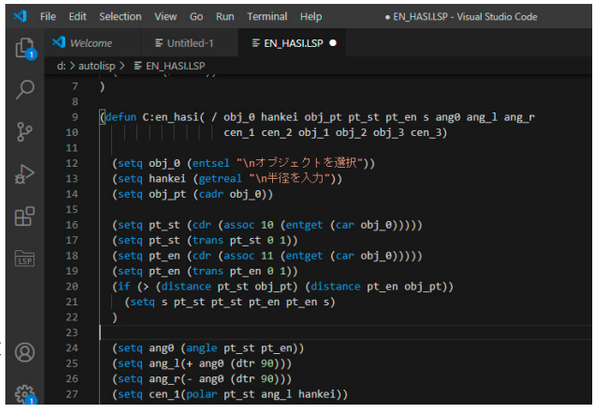
CDL も括弧や変数、文字が色分けして表示されます。
また、コメントは背景色がグレーで表示されます。

4 Microsoft Visual Studio Code

1 Microsoft Visual Studio (VS) コードを利用する (AutoCAD2021 版以降)

システム変数 (LISPSYS) で、AutoLISP の開発環境を変更できます。
LISPSYS <0>・・・・・AutoCAD 付属の VisualLISP エディタ
LISPSYS <1><2>・・・Microsoft 提供の Visual Studio (VS) コード





AutoLISP 開発環境を切り替えるには、システム変数 LISPSYS の現在の値を変更した後、AutoCAD の再起動が必要です。

0	既定のエディタとして Visual LISP IDE (VL IDE) が設定されますが、AutoLISP 関数は Unicode 文字を完全にはサポートしません。 AutoLISP ソース (LSP) ファイルは、保存およびコンパイル時に ASCII (MBCS) 文字セットを使用します。
1	既定のエディタとして Visual Studio (VS) Code が設定され、AutoLISP 関数は Unicode 文字を完全にサポートします。 AutoLISP ソース (LSP) ファイルは、保存時は VS Code のエンコード セットを使用し、コンパイル時は Unicode 文字セットを使用します。
2	既定のエディタとして Visual Studio (VS) Code が設定され、AutoLISP 関数は Unicode 文字を完全にサポートします。 AutoLISP ソース (LSP) ファイルは、保存時は VS Code のエンコード セットを使用し、コンパイル時は ASCII (MBCS) 文字セットを使用します。

AutoCAD 2021 以降は UNICODE をベースとした文字列の扱いになりました。
AutoCAD で文字列を表示したり、AutoLISP のプログラムを書くのも UNICODE を使います。
従前の SHIFT_JIS は扱えません。

UNICODE でプログラミングする場合は、新しいシステム変数 LISPSYS を 1 または 2 にします。
このシステム変数を 0 以外にすると AutoLISP の開発環境が Visual LISP から Visual Studio Code に変更になります。

第 2 節 AutoCAD が提供する AutoLISP

1 ACAD.lsp と ACADDOC.lsp の自動ロードと自動実行

1 AutoCAD の起動時に自動的に図面内に読み込まれる AutoLISP

① ACAD.LSP と ACADDOC.LSP です。この 2 つのファイルは、AutoCAD のバージョンによって、ACAD20**.LSP または ACAD20**.LSP となっています。

② ACAD.LSP ファイル（AutoCAD が起動した直後に、一度だけロードされます。）
特定の AutoLISP プログラムを常に使用する場合は、acad.lsp ファイルに記述します。
AutoCAD を起動すると、サポート ファイル検索パスを使用して acad.lsp ファイルが検索されます。acad.lsp ファイルが見つかると、メモリにロードされます。

③ ACADDOC.LSP ファイル（図面を開くごとに、ロードされます。）
acaddoc.lsp ファイルは、各ドキュメント (図面) の初期化に使用することを目的としています。このファイルは、新しい図面を開始するたびに (または、既存の図面を開くたびに) 使用できる AutoLISP プログラムのライブラリをロードしたいときに使用します。

④既定では、acad.lsp ファイルは AutoCAD の起動時に 1 回しかロードされませんが、acaddoc.lsp ファイルはそれぞれのドキュメント (図面) が開かれるたびにロードされます。
つまり、acad.lsp ファイルの読み込みはアプリケーションの起動ルーチンの一部であり、acaddoc.lsp ファイルの読み込みはドキュメントの初期化処理の一部であると言えます。
また、これらの起動ファイルの既定の動作は、システム変数 ACADLSPASDOC の設定により変更できます。

acad.lsp の初期値は <0> ですが、システム変数 (acadlspasdoc) で変更できます。

システム変数（acadlspasdoc）	
acad.lsp ファイルをすべての図面にロードするか、セッションで開かれる最初の図面だけにロードするかをコントロールします。（初期値 : 0）	
0	セッションで開かれた最初の図面にのみ acad.lsp をロードします。
1	図面が開かれるたびに acad.lsp をロードします。

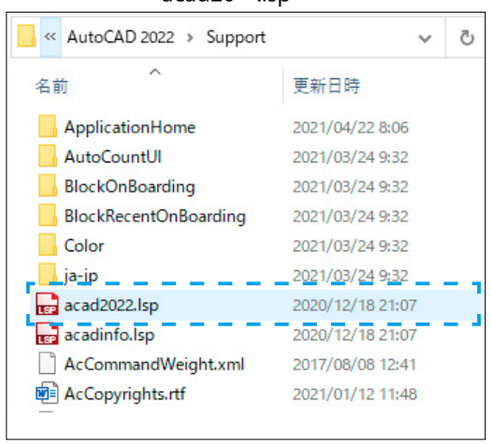
実行可能ファイルのロード元を信頼できるフォルダのみに制限するシステム変数。

システム変数（secureload）	
0	警告を表示せずに、すべての場所から実行可能ファイルをロードします。（非推奨）
1	実行可能ファイルが、システム変数 TRUSTEDPATHS で指定された信頼する場所にある場合にのみロードします。
2	実行可能ファイルの場所が、システム変数 TRUSTEDPATHS で指定されている場合にのみロードすることができます。

2 2 つの LSP の保存場所と内容

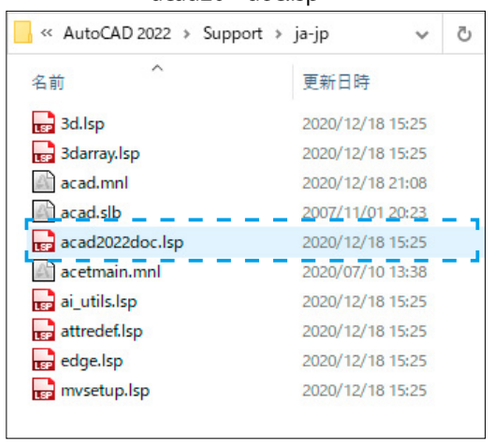
起動時や図面を開くたびにロードされる 2 つの LISP プログラム（acad20**.lsp と acad20**doc.lsp）は、テキスト ファイルですからテキスト エディタで確認できます。
インストールされている場所は、初期値では (Program Files¥Autodesk¥ACAD 20**¥support) にあります。(バージョンによって変わる場合があります。)

acad20**.lsp



Autodesk\AutoCAD 20**\Support\

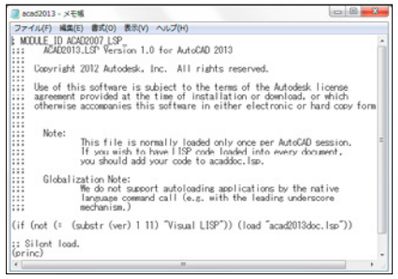
acad20**doc.lsp



Autodesk\AutoCAD 20**\Support\ja-jp\

ユーザーが作成した AutoLISP は、この lisp に追加することは可能ですが、AutoCAD が提供する acad.lsp と acdadoc.lsp の内容を編集することはできません。

acad20**.lsp



acad.lsp の内容の一部抜粋

```
(if (not (=  (substr (ver) 1 11) "Visual LISP"))
(load "acad20**doc.lsp"))

;; Silent load.
(princ)
```

acad.lsp にユーザー定義を記述した例 (P1-79)

acad20**doc.lsp

acaddoc.lsp の内容の一部抜粋

```
(defun ai_AppLoaded (appname apptype)
  (apply 'or
    (mapcar
      '(lambda (j)
          (wcmatch
        (strcase j T)
```

acaddoc.lsp にユーザー定義を記述した例 (P1-79)

2 ACAD.lsp と ACADDOC.lsp の相違

1 ACAD.LSP

AutoCAD が提供する ACAD.LSP（最初の図面でのみ有効 < 初期値 >）

①特定の AutoLISP プログラムを使用する場合は、acad.lsp ファイルに追加して記述します。AutoCAD を起動すると、サポート ファイル検索パスを使用して acad.lsp ファイルが検索されます。acad.lsp ファイルが見つかった場合、それがメモリにロードされます。

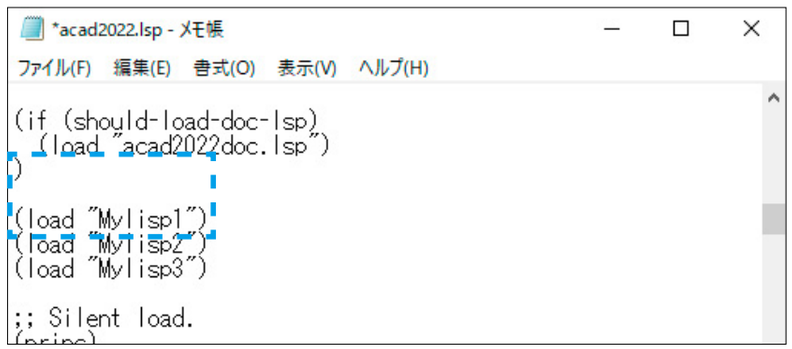
② AutoCAD を起動すると、各図面セッションが開始されるたびに acad.lsp ファイルがロードされます。 acad.lsp ファイルは、アプリケーション固有の起動ルーチン用に使用することを目的としているため、acad.lsp ファイル内で定義されているすべての関数と変数は、最初の図面でのみ有効です。すべての図面で使用できるようにするには、そのルーチンを acad.lsp ファイルから acaddoc.lsp に移動します。

③ acad.lsp と acaddoc.lsp の機能は、システム変数 ACADLSPASDOC で変更できます。システム変数 ACADLSPASDOC を 0(既定) に設定すると、acad.lsp ファイルがアプリケーション起動時に 1 度だけロードされます。ACADLSPASDOC を 1 に設定すると、acad.lsp ファイルは新しい図面を開始するたびに再ロードされます。

④ acad.lsp ファイルには、1 つ以上のルーチンの AutoLISP プログラムを格納できますし、一連の load 関数の呼び出しだけを格納することもできます。後者の方法の方が修正が容易になります。

次のコードを acad.lsp ファイルとして保存すると、各図面セッションが開始されるたびに MyLISP1.lsp、MyLISP2.lsp、MyLISP3.lsp の各ファイルがロードされます。

(load "MyLISP1")
(load "MyLISP2")
(load "MyLISP3")



予約済みの acad.lsp ファイルは変更しないでください。Autodesk が提供する acad.lsp ファイルには、AutoCAD が必要とする AutoLISP 定義関数が含まれています。このファイルは、acad.lsp ファイルがロードされる直前にメモリにロードされます。

2 ACADDOC.LSP

AutoCAD が提供する ACADDOC.LSP（すべての図面で有効）

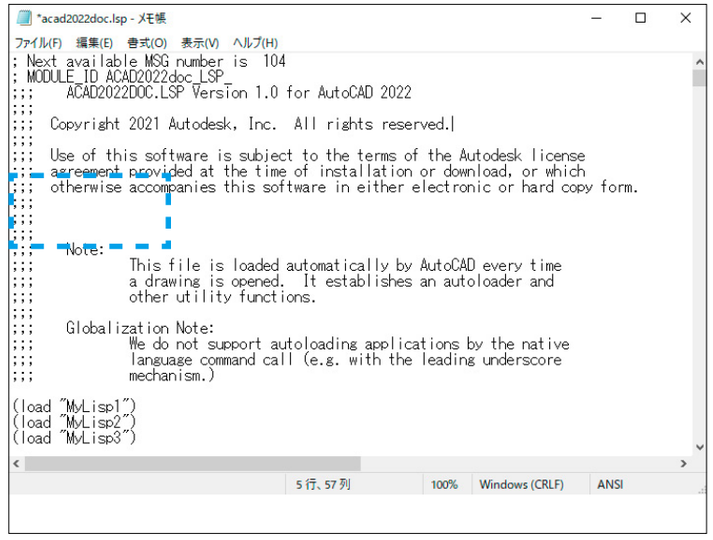
①acaddoc.lsp ファイルは、各ドキュメント (図面) の初期化に使用することを目的としています。このファイルは、新しい図面を開始するたびに (または、既存の図面を開くたびに) 使用できる AutoLISP ルーチンのライブラリをロードします。

② AutoCAD を起動すると、プログラムは、ライブラリ パスで acaddoc.lsp ファイルを検索します。このファイルが見つかると、メモリにロードされます。acaddoc.lsp ファイルは、システム変数 ACADLSPASDOC の設定に関係なく、常に各図面に対してロードされます。

③ユーザーは、すべての AutoLISP ルーチンに対して 1 つの acaddoc.lsp ファイルを持ちます。AutoCAD は、ライブラリ パスで定義されている順序で acaddoc.lsp ファイルを検索します。したがって、この機能を使用すると、各図面フォルダ内に異なる acaddoc.lsp ファイルを格納でき、特定の種類の図面や特定の作業用に特定の AutoLISP プログラムをロードすることができます。

④ acaddoc.lsp ファイルには、1 つ以上のルーチンの AutoLISP プログラムを格納できますし、一連の load 関数の呼び出しだけを格納することもできます。後者の方法の方が、修正が容易になります。次のコードを acaddoc.lsp ファイルとして保存すると、新しいドキュメントが開かれるたびに MyLISP4.lsp、MyLISP5.lsp、MyLISP6.lsp の各ファイルがロードされます。

(load "MyLISP4")
(load "MyLISP5")
(load "MyLISP6")



予約済みの acaddoc.lsp ファイルは変更しないでください。Autodesk が提供する acaddoc.lsp ファイルには、AutoCAD が必要とする AutoLISP 定義関数が含まれています。このファイルは、acaddoc.lsp ファイルがロードされる直前にメモリにロードされます。

2 ACAD.lsp と ACADDOC.lsp の相違

1 ACAD.LSP

AutoCAD が提供する ACAD.LSP（最初の図面でのみ有効 < 初期値 >）

①特定の AutoLISP プログラムを使用する場合は、acad.lsp ファイルに追加して記述します。AutoCAD を起動すると、サポート ファイル検索パスを使用して acad.lsp ファイルが検索されます。acad.lsp ファイルが見つかった場合、それがメモリにロードされます。

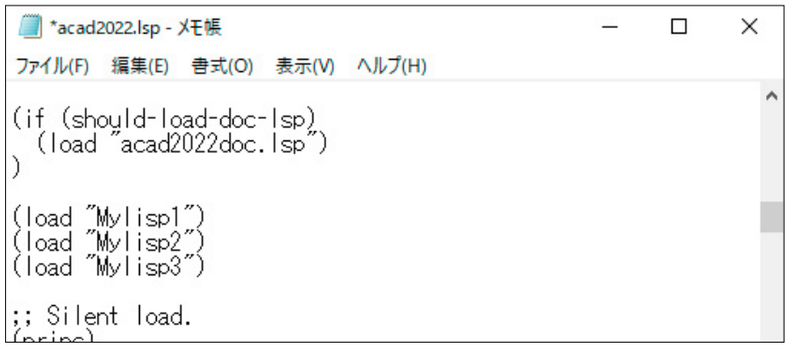
② AutoCAD を起動すると、各図面セッションが開始されるたびに acad.lsp ファイルがロードされます。 acad.lsp ファイルは、アプリケーション固有の起動ルーチン用に使用することを目的としているため、acad.lsp ファイル内で定義されているすべての関数と変数は、最初の図面でのみ有効です。すべての図面で使用できるようにするには、そのルーチンを acad.lsp ファイルから acaddoc.lsp に移動します。

③ acad.lsp と acaddoc.lsp の機能は、システム変数 ACADLSPASDOC で変更できます。システム変数 ACADLSPASDOC を 0(既定) に設定すると、acad.lsp ファイルがアプリケーション起動時に 1 度だけロードされます。ACADLSPASDOC を 1 に設定すると、acad.lsp ファイルは新しい図面を開始するたびに再ロードされます。

④ acad.lsp ファイルには、1 つ以上のルーチンの AutoLISP プログラムを格納できますし、一連の load 関数の呼び出しだけを格納することもできます。後者の方法の方が修正が容易になります。

次のコードを acad.lsp ファイルとして保存すると、各図面セッションが開始されるたびに MyLISP1.lsp、MyLISP2.lsp、MyLISP3.lsp の各ファイルがロードされます。

(load "MyLISP1")
(load "MyLISP2")
(load "MyLISP3")



予約済みの acad.lsp ファイルは変更しないでください。Autodesk が提供する acad.lsp ファイルには、AutoCAD が必要とする AutoLISP 定義関数が含まれています。このファイルは、acad.lsp ファイルがロードされる直前にメモリにロードされます。

2 ACADDOC.LSP

AutoCAD が提供する ACADDOC.LSP（すべての図面で有効）

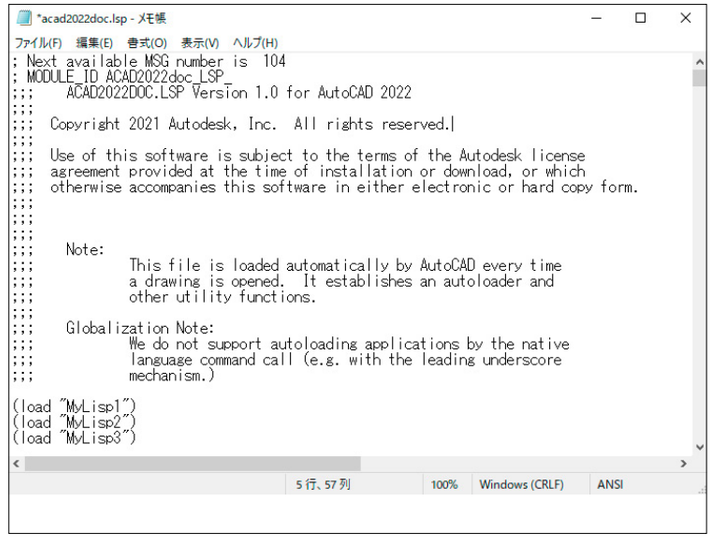
①acaddoc.lsp ファイルは、各ドキュメント (図面) の初期化に使用することを目的としています。このファイルは、新しい図面を開始するたびに (または、既存の図面を開くたびに) 使用できる AutoLISP ルーチンのライブラリをロードします。

② AutoCAD を起動すると、プログラムは、ライブラリ パスで acaddoc.lsp ファイルを検索します。このファイルが見つかると、メモリにロードされます。acaddoc.lsp ファイルは、システム変数 ACADLSPASDOC の設定に関係なく、常に各図面に対してロードされます。

③ユーザーは、すべての AutoLISP ルーチンに対して 1 つの acaddoc.lsp ファイルを持ちます。AutoCAD は、ライブラリ パスで定義されている順序で acaddoc.lsp ファイルを検索します。したがって、この機能を使用すると、各図面フォルダ内に異なる acaddoc.lsp ファイルを格納でき、特定の種類の図面や特定の作業用に特定の AutoLISP プログラムをロードすることができます。

④ acaddoc.lsp ファイルには、1 つ以上のルーチンの AutoLISP プログラムを格納できますし、一連の load 関数の呼び出しだけを格納することもできます。後者の方法の方が、修正が容易になります。次のコードを acaddoc.lsp ファイルとして保存すると、新しいドキュメントが開かれるたびに MyLISP4.lsp、MyLISP5.lsp、MyLISP6.lsp の各ファイルがロードされます。

(load "MyLISP4")
(load "MyLISP5")
(load "MyLISP6")



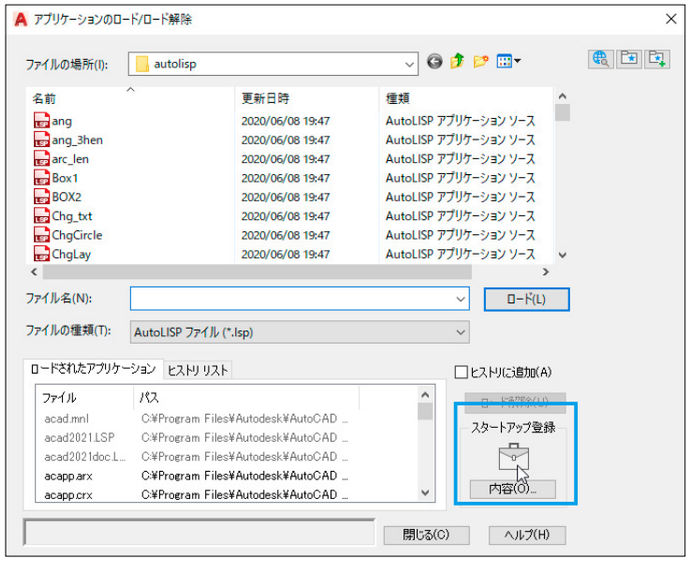
予約済みの acaddoc.lsp ファイルは変更しないでください。Autodesk が提供する acaddoc.lsp ファイルには、AutoCAD が必要とする AutoLISP 定義関数が含まれています。このファイルは、acaddoc.lsp ファイルがロードされる直前にメモリにロードされます。

3 APPLOAD[アプリ ロード]

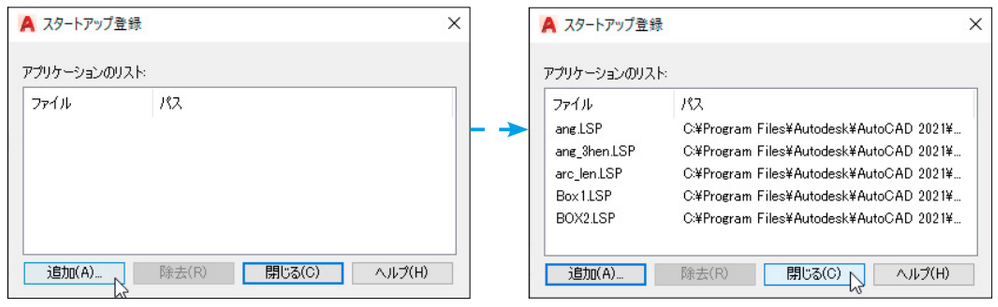
1 常時使用する

① AutoLISP を常時使用するには、APPLOAD[アプリ ロード] コマンドを使い、表示される [アプリケーションのロード / ロード解除] パネルにある [スタートアップ登録] で指定します。

② [アプリケーション] パネル → [アプリロード] を使います。
[アプリケーションのロード / ロード解除] パネルの [スタートアップ登録] で常時使用する LISP を指定します。

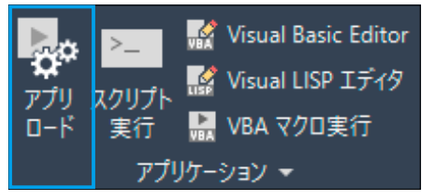


③ [スタートアップ登録] の [追加] ボタンを押して、使用する LISP を指定します。
どの図面に対しても、LISP は使用できます。

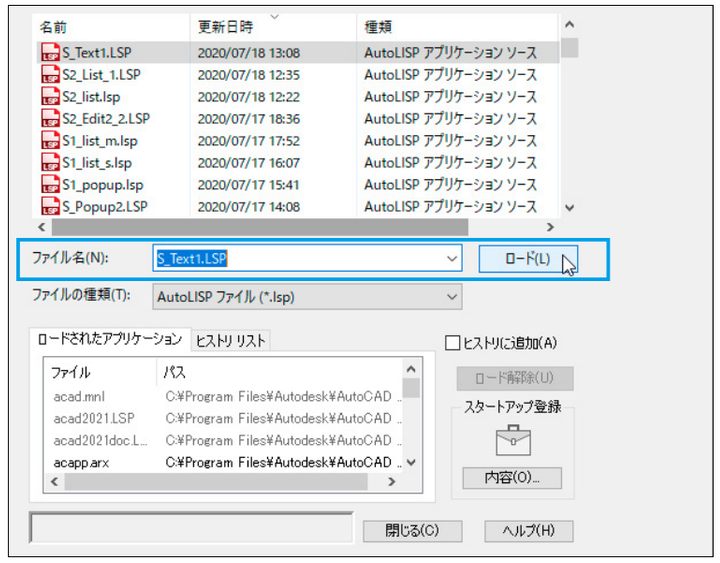


2 一時的に使用する

① AutoLISP を一時的に使用するには、APPLOAD[アプリ ロード] コマンドを使い、表示される [アプリケーションのロード / ロード解除] パネルに表示される LISP を指定します。

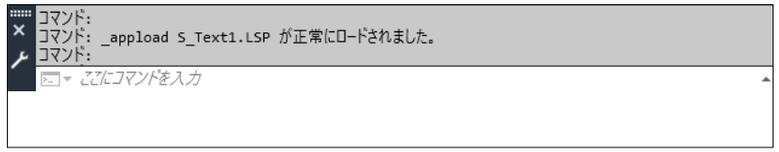


② [アプリケーション] パネル → [アプリロード] を使います。
[アプリケーションのロード / ロード解除] パネルに表示される AutoLISP を選択して、[ロード] ボタンを押します。

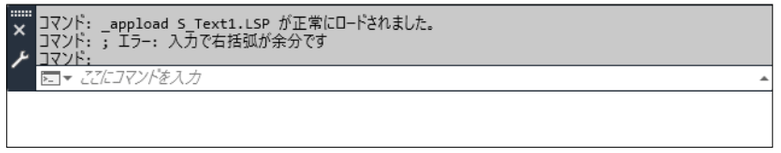


③ロードした AutoLISP は、その図面内でのみ有効です。
新規の図面や他の図面で使用するためには、再度 AutoLISP をロードする必要があります。

④正常にロードされた時は、コマンドラインに通知されます。



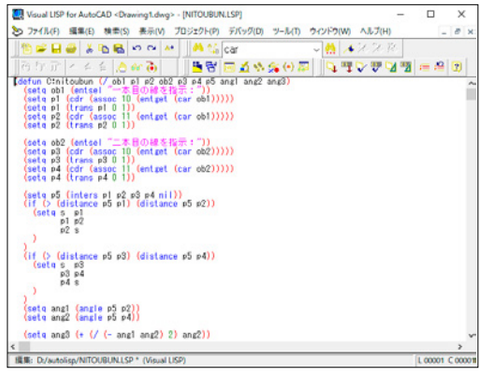
もし、LISP に誤りがあるときは、その旨通知されますが、誤りの箇所はわかりません。



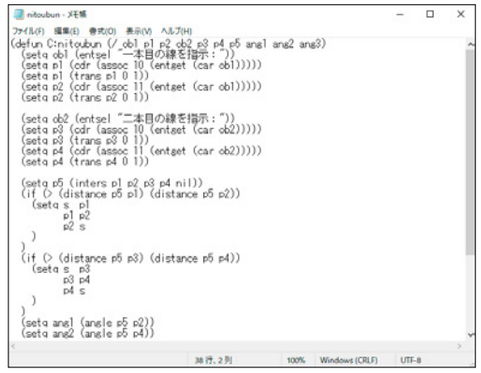
第 3 節　AutoLISP の作成から実行まで

[ツール] → [AutoLISP] → [Visual LISP エディタ] を用いて作成しますが、Windows 付属の [メモ帳] や [ワードパット] などのテキスト エディタでも作成できます。

Visual LISP エディタで記述



メモ帳で記述



1 AutoLISP の書式

```
(defun C:Myline( / p1 p2)
        (setq p1 (getpoint "¥n 始点を指示："))
        (setq p2 (getpoint p1 "¥n 終点を指示："))
        (command "LINE"  p1 p2 "")
        (princ)
)
```

最初の左括弧 < (> から、最後の右括弧 <) > までが、1 つのプログラムになります。

defun・・・関数の宣言です。

C :　・・・関数名（この場合は <MyLine>）の前に C: を付けると、自分の関数を AutoCAD のコマンドと同じ様に使うことができます。
もし、付けない場合 <defun MyLine()> では、コマンドラインから <(MyLine)> のように括弧でくくって呼び出さないといけません。

MyLine・・自分が作成した関数です。拡張子の lsp は必要ありません。

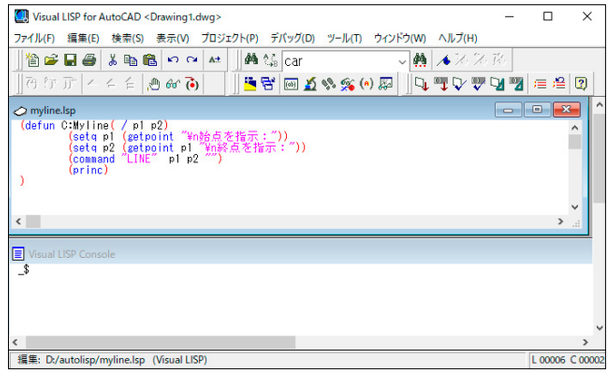
（　）　・・変数を記述します。強制ではありませんが、空白にして変数を使用した場合は、プログラムが終了した後でも、変数がメモリーに残ります。
ただし、変数が無くても ( ) は必要です。

☞defun<declare function>

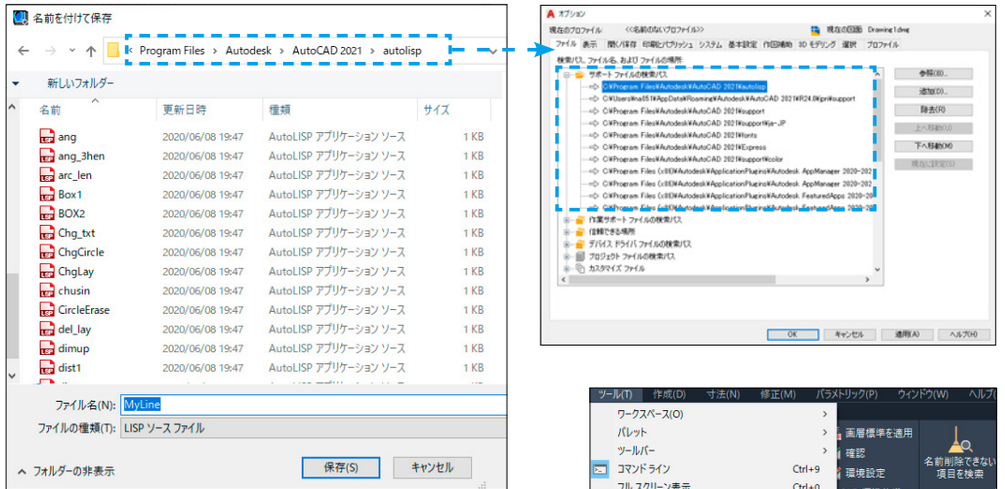
2 AutoLISP の作成から読み込み

線分を描くプログラムを作成して、LISP プログラムの作成から実行までの流れは以下の通りです。

① Visual LISP エディタを開き、線分プログラム (MyLine.lsp) を作成します。
（拡張子は、lsp になります。）

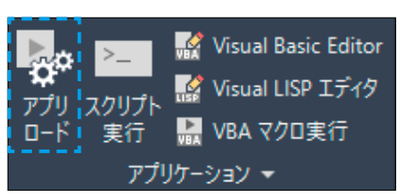


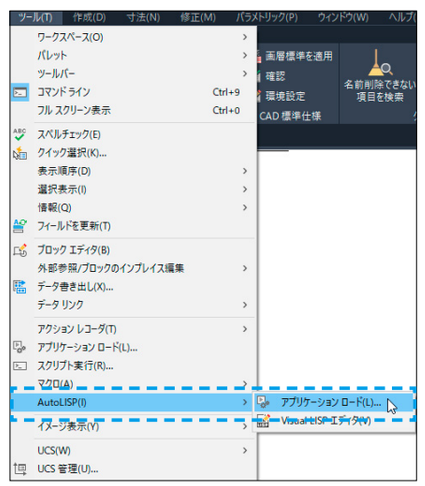
②このファイルの名前を <MyLine.lsp> にして、AutoCAD と同じフォルダに保存します。
または、保存したフォルダに AutoCAD のサポート ファイルの検索パスを通します。



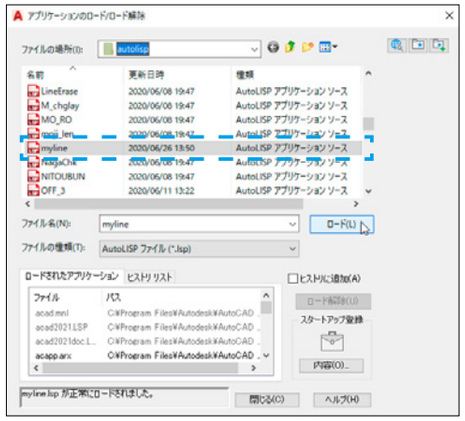
③プルダウンメニューから [ツール] → [AutoLISP] → [アプリケーション ロード] を選択します。

④又は、[管理] タブ→ [アプリケーション] パネル → [アプリロード] を選択します。

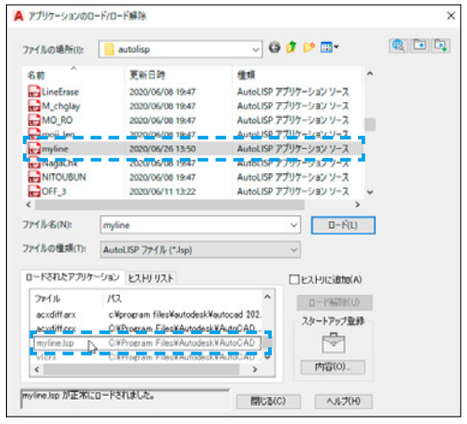




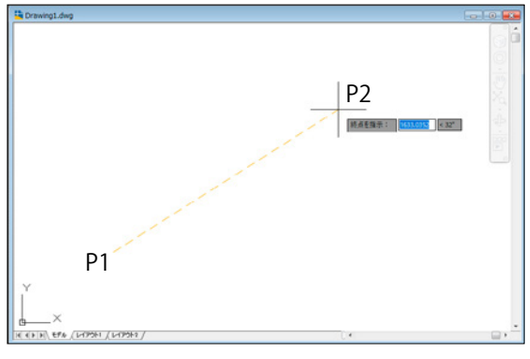
④ [アプリケーションのロード / ロード解除] ダイアログから <MyLine.lsp> を選択します。
[ロード] ボタンを押して、メモリに読み込みます。

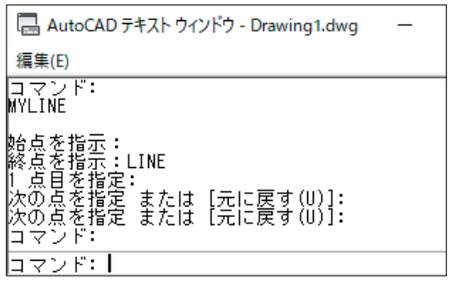


⑤正常に読み込まれた LISP ファイルは、[ロードされたアプリケーション] の中に登録されています。
この中にロードされた LISP は、キーボードから LISP 名を入力することによって使用できます。



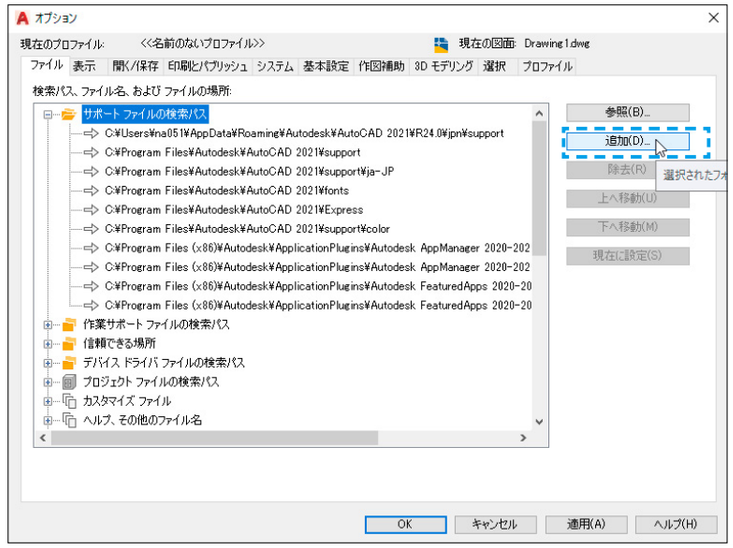
⑥ AutoCAD のコマンドラインから、<MyLine> と入力します。(拡張子の lsp は要りません。)
コマンドラインに < 始点を指示 : > のメッセージが出ます。
マウスで点 P1 を指示すると、次に < 終点を指示 : > のメッセージが出ます。
終点 (P2) を指示すると、点 P1 と点 P2 間に線分が作図されます。



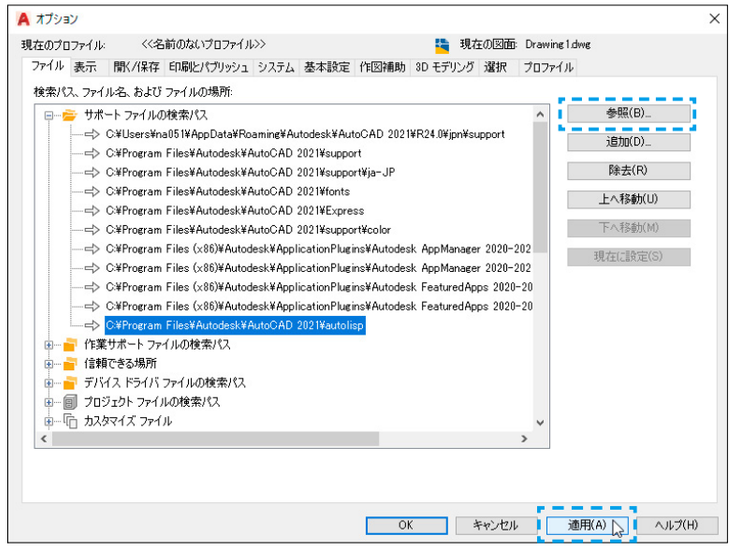


3 サポート ファイルの検索パスを作成する

① [ツール] → [オプション] ダイアログを開きます。
[ファイル] タブの [サポート ファイルの検索パス] をクリックします。



② [追加] ボタンを押します。
ユーザー専用のファイル保存フォルダを指定します。
(指定するフォルダは予め作成しておきます。)



プログラムのメンテナンスを考慮すると、AutoCAD のフォルダとは別のフォルダを作成し、プログラム専用のフォルダにした方が良いでしょう。

第 4 節　AutoLISP の式

AutoLISP では、ユーザー自身が関数を定義できます。一度定義すると、それらの関数を標準の関数を使用するのと同じように AutoCAD のコマンド ラインや AutoLISP の他の式で使用できます。コマンドは関数の特殊な形式にすぎないので、ユーザー独自の AutoCAD コマンドを作成することができます。

一般的に AutoCAD が提供する関数を組み込み関数と呼ばれ、ユーザーが作成した AutoLISP などの関数はユーザー関数とか補助関数と呼ばれています。
ユーザー関数の書式は次のように、関数名の前に <C:> を付加します。
　(defun　C:MyLisp( ))・・・ユーザー関数

また、補助関数の書式では次のように、関数名の前には <C:> は必要ありません。
　(defun　MyLISP( ))　・・・補助関数

このユーザー関数や補助関数では、関数名の前に defun<declare function> を付けて、LISP 関数であることを宣言します。
<C:> を付けたユーザー関数と付けない補助関数の呼び出し方の違いを以下に示します。

1　ユーザー関数と補助関数の呼び出し方

DEFUN	関数を定義します。

AutoCAD のコマンド名を定義します。C: を付けると AutoCAD と同じように使用出来ます。
C: を付けない場合は、コマンドラインからの入力時に関数名を ( ) で囲みます。

① (defun C: MyLisp ( )
　　・・・・・
　)　→ コマンド ラインから、MyLisp と入力します。

② (defun　MyLisp ( )
　　・・・・・
　)　→ コマンド ラインから、(MyLisp) と入力します。

メモリーに読み込む時は、どちらも (load "MyLisp") ですが、
実行時に ( ) を付けるか付けないかの違いです。

2　AutoLISP の処理を確認する

コマンドラインからでも、LISP は使えます。
キーボードから次のように入力します。　→　（+ 3 2）
コマンドラインの次の行にメッセージが表示されます。　→　5

このように加算は < + > ですが、引き算は < - >、掛け算は < * >、割り算は < / > の記号を使用します。また、式の始めと終わりは必ず（　）で囲まれていなければいけません。

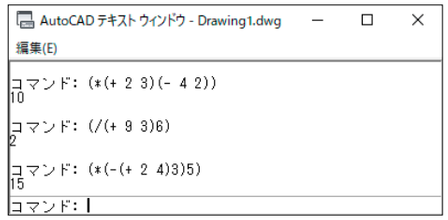
① AutoLISP は、括弧で囲まれた中に関数や数値をスペースで区切って表します。
括弧は式の重要な要素です。すべての括弧は対を構成しなければなりません。
つまり、どの左括弧にもそれに対応する右括弧が必要です。

② AutoLISP の式は、何かの関数 (演算子) とそのあとに計算される項目を含んでいる必要があります。計算される項目を関数の引数とも呼びます。
したがって、(+ 3　2) では、+ が関数、数値 3 と 2 が引数になります。

③上の式で、式の関数と引数との区切りにスペースを使っていることに注意してください。
括弧の中の要素の間にはかならずスペースが必要です。括弧と式の要素の間にはスペースは不要ですが、式が複雑になったときに読みやすくするためにスペースを入れることは構いません。

④ AutoLISP では、括弧の中の 1 番目が関数の名前である場合、これを式と認識してその結果を返します。しかし、1 番目が LISP の関数でない場合は、エラーになります。
（式の中では、単なるリストと認識して、そのまま文字列または数値を返すこともあります。）
（例）(/ 10　2)　→　5
　　(10　2)　→　; エラー : 関数が間違っています :

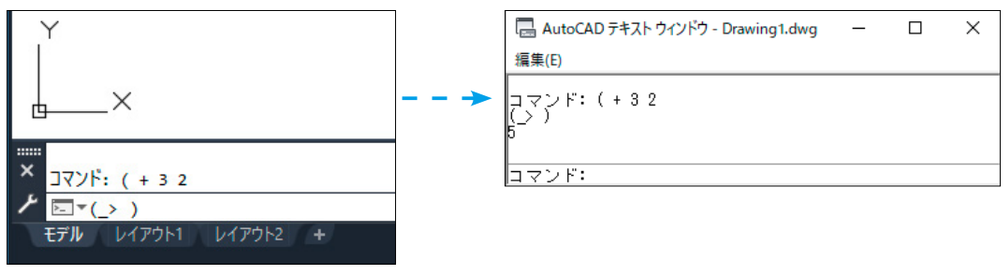
通常の表記		LISP の表記		結果
(2+3)×(4-2)	→	(*(+2 3)(-4 2))	→	10
(9 + 3)÷6	→	(/ (+9 3) 6)	→	2
(2 + 4 - 3) × 5	→	(*(-(+2 4) 3) 5)	→	15



⑤ AutoLISP で一番よく起きるエラーは、括弧の付け忘れです。その場合は、LISP が終わるまでに必要な括弧の数が表示されます。

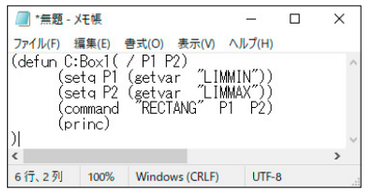
コマンド：(+ 3 2
(_ >　　・・・括弧が 1 つ足りないので) を入力する。
5　　・・・答えが表示される。

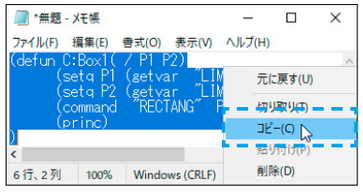
）を 1 つ入力します。　　正しく計算されます。



⑥ 他のファイルからテキストをクリップボードにコピーして、コマンドラインに貼り付けることも可能です。（P1-33 box1.lsp の①から③を貼り付けました。）

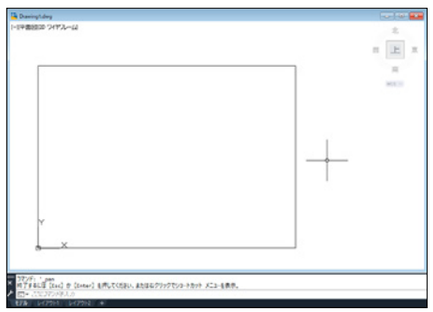
Step-1 （左図）メモ帳に記述した 3 行の文字列。 （右図）マウスで選択して、コピーします。





Step-2　AutoCAD のコマンドラインから、右ボタンを押して [貼り付け] を選びます。

Step-3　貼り付けた文字列を順番に処理して、作図を行います。（長方形が作図されました）



3 変数に値をセットする

setq 関数	変数に値を代入します

変数に値をセットする関数です。同じ機能に <'>(アポストロフィー) があります。
変数 [pt] に値 <10.0> をセットするには、
→ (setq pt 10.0) または、(setq pt '10.0) となります。

①一般的なプログラムでは、変数に値を代入するには、（変数が a、値が 10 の場合）
→ a = 10

② LISP では、setq という関数を使います。上記の例では
→ (setq a 10)

③値が文字 (AutoCAD) の場合は、文字列を <" "> で囲みます。
→ (setq a “AutoCAD”)
このように、AutoLISP の変数には、数値や文字列が代入できます。

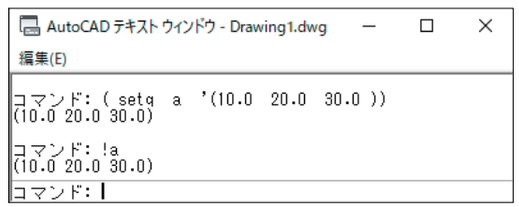
list 関数	任意の数の式を受け取り、それらを 1 つのリストに結合します

複数の要素を 1 つのリストにします。
変数 [pt] に複数の要素 <10.0 20.0> をセットするには、
→ (setq pt (list 10.0 20.0)) または、(setq pt '(10.0 20.0)) となります。

変数に 2 つ以上の値を代入する場合
④ 2 次元の点 (X 座標、Y 座標) では
→ (setq a '(10.0 20. 0)) または、 (setq a (list 10.0 20.0))

⑤代入する値が 3 次元座標 (X,Y,Z) の場合は
→ (setq a '(10.0 20.0 30.0)) または、 (setq a (list 10.0 20.0 30.0))

⑥変数の値を確認するには、 ！（感嘆符）を使います。
キーボードから、(setq a '(10.0 20.0 30.0)) と入力した後に、!a と入力します。
コマンドラインに < (10.0 20.0 30.0)> の数値が表示されます。



4 AutoLISP プログラム ファイル内のコメント

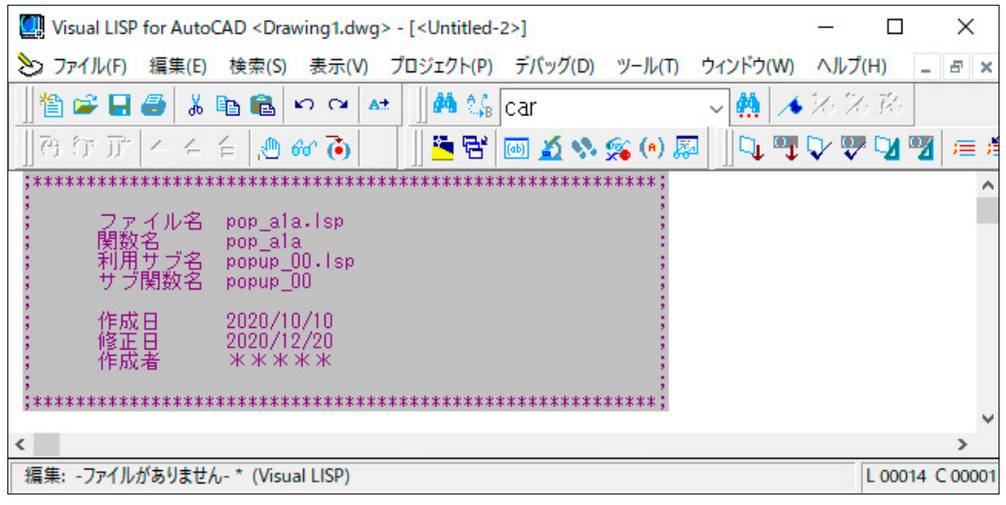
AutoLISP のプログラムにコメントを記述することは大切です。
コメントはプログラムの作成者だけでなく、将来他のプログラマが必要に応じてプログラムを改良する時に有益です。

コメントは以下のような場合に作成します。
①プログラム ファイル名、サブ関数名、DCL ファイル名などを明記する。
② LISP プログラムの使用方法などを明記する。
③ LISP プログラム内に、適宜コメントを挿入する。

コメントは 1 つ以上のセミコロン < ; > で始まり、その行の終わりまで続きます。
改行ごとに、その行の初めにセミコロン < ; > を付けます。

プログラムの中にコメントを入れる場合は、< ;|　|; >（インライン コメント）の形になります。
この中の全ての文字は無視されます。

1 LISP プログラムの名前や関数名、作成者名、作成日等を記述します。
プログラムの先頭に以下のように、判りやすいレイアウトで作成するのが良いでしょう。



< ; > が必要なのは行の先頭だけですから、それ以降は自由にレイアウトを組むことができます。
LISP プログラム全体に同じデザインで統一すれば、他のプログラマが参考にする場合にも判りやすくなります。
関数名以外では、新規作成日や修正の履歴や作成者も記述しておきます。

2 プログラムの使用方法の説明にコメントを付けます。

```
;-------------------------------------------------------------------------------;
;<<<<< リストの処理 >>>>>
; イメージ ファイルとテキスト ファイルを順番に読み込んで、DCL に表示する。
; 各ファイルは、AutoCAD のパスの通ったディレクトリに保存しておくこと。
;-------------------------------------------------------------------------------;
(defun make_list_box(make-image-list file-name-list / rep-num rep-start)
  (start_list make-image-list)
  (setq rep-start 0)
  (setq rep-num (length file-name-list))
  (repeat rep-num
    (add_list (nth rep-start file-name-list))
    (setq rep-start (1+ rep-start))
  )
  (end_list)
```

3 プログラム内の注釈にコメントを付けます。

① 1 行コメント
コメントは 1 つ以上のセミコロン (;) で始まり、その行の終わりまで続きます。

```
(start_image "pop_w1a")
  (fill_image 0 0 x y -2)                    ; タイルを -2 の色で塗り潰し
  (slide_image 0 0 x y (nth image-num sub-name-list))  ; スライドファイルを表示
(end_image)
```

②インライン コメント
(;| ・・|;) 内のすべてのテキストは無視されます。これにより、コメントを 1 行のコード内に入れたり、複数の行に続けることができます。

```
  (command "layer" "on" "g_wan" "s" "g_wan" "")
  (command "circle" a_e 0.00211)
  (command "layer" ;| 画層を初期値にセット |;"s" "0" "")
  (command "layer" "lo" "g_wan" "off" "g_wan" "")
```

第 5 節 AutoLISP のデータタイプ

1 AutoLISP のデータタイプ

AutoLISP で使用される主要なデータ タイプの例を示します。
データをタイプに分けることでデータを正確に評価できて、プログラムを迅速に実行できます。

番号	データ タイプ	例
①	整数	10
	実数	0.314
②	文字列	“abc” 文字列は、“ と “ で囲まれます。
③	リスト	(10.0 20.0 30.0)
④	選択セット	<Selection Set : 1>
⑤	図形名	<Entity name : 60000014c>
⑥	FILE（ファイル ディスクリプタ）	<File : a2165>
⑦	SYM（シンボル）	pt1
⑧	SUBR（定義関数名）	MyLine.lsp

1 整数と実数

整数は、-2,147,483,648 から +2,147,483,647 以内のすべての整数です。整数だけを含む式の値は、常に整数になります。
たとえば、 (/ 25 2) の値は 12 です。12.5 にはなりません。LISP は計算結果の小数部を切り捨てて整数にします。
これに対して、実数は小数部を含む値です（浮動小数点）。上の式を使って (/ 25 2.0) とすると、その値は 12.5 になります。

整数の 12 はつねに 12 です。実数の 12.345 と 12.346 は同じではありませんが、小数点以下の桁を切り捨てれば同じになってしまいます。
整数のこの明快な性質は、個数をカウントするような用途に向いています。一方、実数は座標値や角度などを正確に計算する場合に向いています。

Point!

(* 25 2) → 12

(* 25.0 2) → 12.5

実数の表記では、最初の 0 は省略できません。つまり、0.314 は .314 とはできません。
LISP では、0.1 と -0.1 との間をとる実数は必ず 0 から始まります。

2 文字列

文字列はテキストのことです。文字列は LISP の式のプロンプトによく使われますが、LISP で操作することもできます。

Point!

(prompt “AutoCAD”) → コマンドラインに <AutoCAD> の文字が表示される。

(setq A “LISP”) → 変数 A に LISP の文字がセットされる。

prompt はコマンドラインに文字を表示させる関数です。
setq は次の変数にその後の数値や文字を代入させる関数です。

3 リスト

データ要素を並べて括弧で囲んだものをリストと呼びます。リストは AutoLISP の基本データ構造です。
リストには、整数、実数、文字列などがいくつあっても、また混在していても構いません。
また、リストの中にリストが入っていても（入れ子）構いません。

リストには 2 つのタイプがあります。
1 つは、計算を目的とするもの。もう 1 つは、複数のデータ（要素）を組み合わせて 1 つのセットにすることを目的とするものです。この代表には、X,Y,Z の座標値があります。

Point!

(+ 2 3) → 括弧の中が計算されて、5 が返される。

(5.0 10.0 15.0) → 最初の 5.0 が X 座標、二番目の 10.0 が Y 座標、最後の 15.0 が Z 座標の組み合わせになる。

4 選択セット

選択セットは、複数の図形（エンティティ）の集まりのことです。
対話形式で、選択セットに図形を追加したり、削除することができます。

Point!

直前に選択された選択セット（選択された図形群）を、シンボル ss1 へ割り当てます。
(setq ss1 (ssget "P")) → 返り値 <SelectionSet 1>

5 図形名

AutoCAD の図形 (オブジェクト) を基本図形 (エンティティ) と呼んでいます。
この基本図形は、作図コマンド (Line や Arc) によって描かれて、個別にユニークな識別番号が与えられます。
この名前は英数字のコードで、図面内でその図形 (オブジェクト) をユニークに識別するものです。
この名前を使って AutoLISP はオブジェクトを選択し、処理しています。
基本図形名は、AutoCAD が自動的に割り当てるもので、ユーザーが決めることはできません。
また、ファイルが変われば、基本図形名も変わります。

Point!

図形を選択する関数 (entsel) で図形を指示すると、下記のような情報がコマンドラインに表示されます。この中の < > 内が図形名です。

(<Entity name : 60000022> (50.0 50.0 0.0)
図形名　　指示した座標

6 FILE（ファイル ディスクリプタ）

AutoLISP は、ディスクのテキストファイルを読み書きできます。
ファイル ディスクリプタ（記述子）は、あらかじめ開いておいたファイルにプログラムからアクセスするときに使用します。つまり、ファイル ディスクリプタ（記述子）とは目的のファイルを示す変数と考えられます。

Point!

(setq text1 (open File_A “r”)) → テキストファイル (File_A) を読み込みモードで開いて、変数 text1 にセットする。

(close text1) → オープンしているファイル (text1) を閉じる。最後は、必ずこの記述が必要。

7 シンボル

AutoLISP では、シンボルを用いて値を格納します。
シンボル名は大文字と小文字を区別せず、下記を除く任意の英数字及び特殊記号の組み合わせを使用できます。

シンボル名で使用できない文字	
(	(左括弧)
)	(右括弧)
.	(ピリオド)
'	(アポストロフィ)
"	(クォーテーション)
;	(セミコロン)

8 SUBR< 定義関数名 >

SUBR は AutoLISP 関数またはコンパイル済み関数です。
これらの関数には、四則計算のような単純な演算子から、指定するオブジェクトの情報を図面データベースから取得するための複雑なものまであります。

Point!

defun またはロード時に指定した名前を以下に示します。

SUBR → 組み込みのコンパイル済み関数
EXRXSUBR → 外部 ARX 関数
USUBR → ユーザー定義関数

データ タイプを確認する

TYPE　変数のデータ タイプを確認します。

① (setq pt (list 10 20 30))
→ コマンド ラインに !pt と入力すると、(10 20 30) が表示されます。
→ コマンド ラインに (TYPE pt) と入力すると、LIST が表示されます。

② (setq text1 "AutoLISP")
→ コマンド ラインに !text1 と入力すると、"AutoLISP" が表示されます。
→ コマンド ラインに (TYPE text1) と入力すると、STR が表示されます。

TYPE 関数が返すデータ タイプを以下の表に示します。

返り値	データ タイプ
ENAME	図形名
EXRXSUBR	外部 ObjectARX アプリケーション
FILE	ファイル ディスクリプタ
INT	整数
LIST	リスト
PAGETB	関数ページング テーブル
PICKSET	選択セット
REAL	実数
SAFEARRAY	セーフ配列
STR	文字列
SUBR	LISP ソース ファイルからロードされたユーザー定義関数
SYM	シンボル
VARIANT	バリアント
VLA-object	ActiveX オブジェクト
nil	値が無い

第 6 節　AutoLISP の変数

1　ローカル変数とグローバル変数

ローカル変数は、1 つの関数内だけに使用され、プログラムが終了するとその値は無くなり (nil) ます。
グローバル変数は、定義関数の外でも有効です。そのため、関数間でその値を共有できます。

AutoLISP における、ローカル変数とグローバル変数の関係は次のようになります。

```
(defun C:MyLisp ( / a )
      (setq  a  3)
      (setq  b  ( +  a  3))
)
```

上記の関数では、変数 <a> は関数の内部でしか使用出来ませんが（ローカル変数）、
変数 <b> は、この関数が終了した後でもメモリに残ります（グローバル変数）。

もし、変数 <b> もローカル変数にするには、

```
①  (defun C:MyLISP ( / a b)                    ← 最初にローカル変数として宣言します。
       ・
       ・
    )
②  (defun C:MyLISP ( / a )
       ・
       ・
     (setq  b  nil)                            ← プログラムの最後に、<nil> をセットします。
    )
```

Point!

Common LISP の慣習で、グローバル変数を明示するために、
変数名の前後に <*(アスタリスク)> を追加することもあります。
→　(setq *b * (+ a 3))
このように、グローバル変数名とローカル変数名の名前の付け方を
決めておいた方が良いでしょう。

2　引数とローカル変数

引数は別の関数から特定の値を持った変数を関数内に取り込みます。
従って、引数は [受け取る変数] とも言えます。
それに対して、ローカル変数を [与える変数] とも言えます

AutoLISP における、引数とローカル変数の関係は次のようになります。

```
(defun MyLisp ( 引数 < 与える変数 > / ローカル変数 < 受け取る変数 > )
        ・
        ・
)
```

下記のプログラム (test2.lsp) には、補助関数 (test1) を含んでいます。
この中で、変数 (a と b) は引数、変数 (c) はローカル変数です。
関数 test1 は変数 a と b の乗算だけです。test1 に 2 つの数値を与えれば、乗算して返してくれる関数です。
関数 test2 はローカル変数 (c) を持ちます。

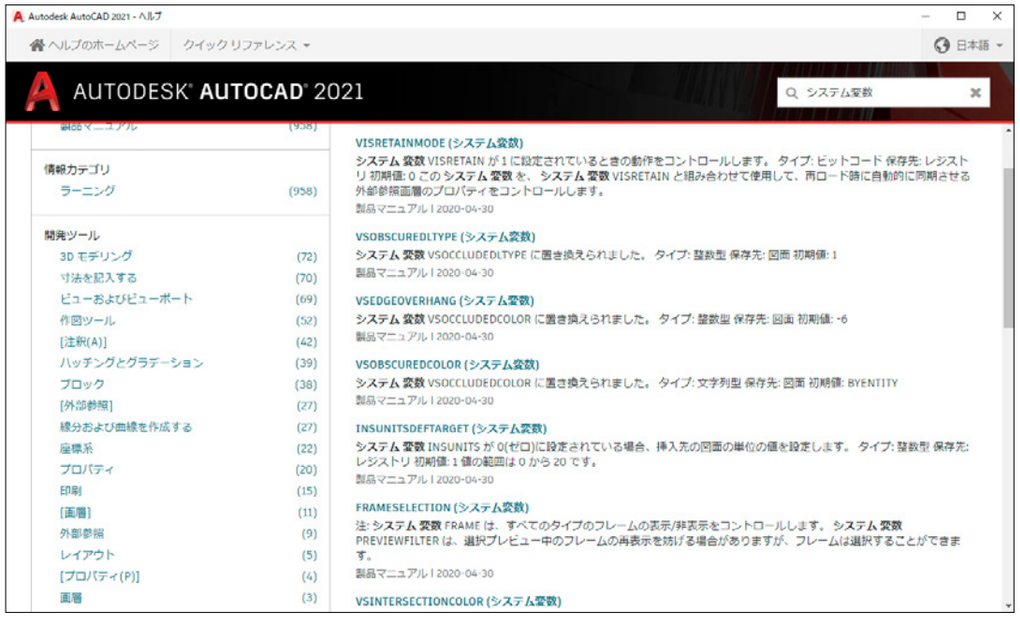
```
(defun test1 (a b)
 (* a b)
)
(defun C:test2( / c)
 (setq c (test1 2 3))
)
```

関数 test2 を実行すると
→ 6 が返ります。
→ !a と入力すると <nil> が返ります。
→ !b と入力すると <nil> が返ります。
→ !c と入力すると <nil> が返ります。

結果として、ローカル変数 (c) だけでなく、引数の (a と b) の値も <nil> になっています。
これから、引数 (a と b) は特殊なローカル変数と言えます。

3 システム変数

getvar 関数と setvar 関数を使用すると、AutoCAD のシステム変数の値を調べたり変更することができます。これらの関数は、変数名を文字列で指定します。setvar 関数では、システム変数に応じたタイプの値を指定します。AutoCAD の システム変数には、整数、実数、文字列、2D 点、3D 点といったさまざまなタイプがあります。setvar に引数として指定する値は、このいずれかのタイプです。



GETVAR	システム変数の値を取得します。

システム変数 [CMDDIA] の現在の値を取得します。

```
(setq  a (getvar  "cmddia"))
```

コマンド : !a　　→ コマンドラインに !a と入力します。
1　　→ コマンドラインに <1> が表示されます。
（1 = PLOT コマンドでダイアログ ボックスを使用します。）

SETVAR	システム変数に値を代入します。

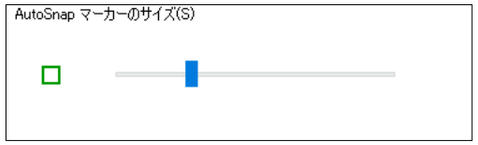
システム変数 [CMDDIA] を <0> にします。

```
(setvar  "cmddia"  0)
```

（ 0 = PLOT コマンドでダイアログ ボックスを使用しません。）

表示関係のシステム変数			
システム変数	種類	保管先	関連コマンド
APERTURE	整数	ACAD.CFG	OSNAP

オブジェクト スナップ範囲の大きさをピクセル単位で指定します。（1 ～ 50）



BLIPMODE	整数	図面	

マーカー（図形や座標点を指示したときに画面上に残る点）の表示を制御します。
0 = オフ（マーカー表示しない < 既定値 >）
1 = オン（マーカー表示する）

CMDECHO	整数		

AutoLISP の COMMAND 関数を使用する時、
0 = エコーバックしません。
1 = プロンプトとユーザーの入力をエコーバックします。< 既定値 >

CMDDIA	整数	ACAD.CFG	PLOT

0 = PLOT コマンドでダイアログ ボックスを使用しません。
1 = PLOT コマンドでダイアログ ボックスを使用します。< 既定値 >

DRAGMODE	整数	図面	

図形を画面上で動的に表示します。
0 = ドラッグ モード <ON>。 ドラッグ機能が有効です。
1 = ドラッグ モード <OFF>。 ドラッグ機能が無効です。
2 = 自動。　　ドラッグ機能が自動的に働きます。< 既定値 >

FILEDIA	整数	ACAD.CFG	

0 = ファイル ダイアログ ボックスを表示しない。
1 = ファイル ダイアログ ボックスを表示する。< 既定値 >

FILLMODE	整数	図面	SOLID、PLINE、TRACE、DONUT

塗り潰しの表示を制御します。（再作図で更新）
0 = オフ（塗り潰しません）
1 = オン（塗り潰します < 既定値 >）

HIGHLIGHT	整数		

図形を選択した時の、ハイライト表示を制御します。
0 = オフ（選択した図形はハイライトしません）
1 = オン（選択した図形はハイライトします < 既定値 >）

MIRRTEXT	整数	図面	MIRROR

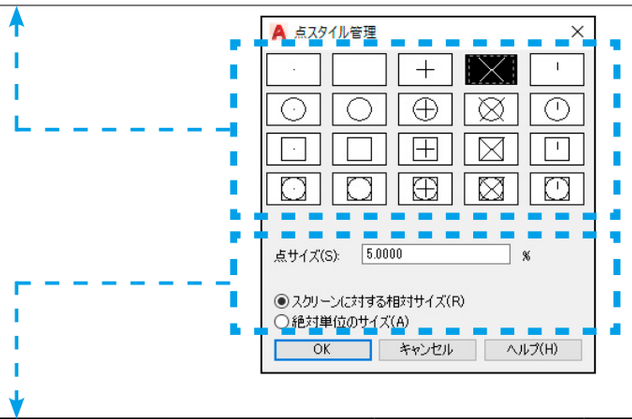
0 = MIRROR コマンドを実行したとき、文字の表示方向は変更しません。< 既定値 >
1 = MIRROR コマンドを実行したとき、文字を鏡像表示します。

MIRRTEXT = 0

MIRRTEXT = 1

PDMODE	整数	図面	POINT、DIVIDE、MEASURE

点の表示モードを指定します。

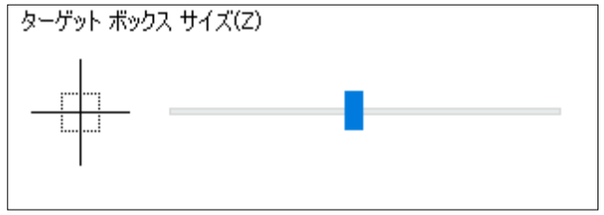


PDSIZE	整数	図面	POINT、DIVIDE、MEASURE

点の表示サイズを指定します。

PICKBOX	整数	ACAD.CFG	

図形を指示するときのボックス カーソルの大きさをピクセル単位で指定します。



PLINEWID	実数	図面	PLINE

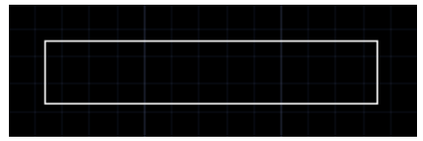
ポリラインの幅を指定します。既定値は <0>

QTEXTMODE	整数	図面	

文字の省略モードを制御します。（再作図で更新されます。）
0 = オフ　　文字省略モードをオフにします。文字が表示されます。< 既定値 >
1 = オン　　文字省略モードをオンにします。文字の代わりにボックスが表示されます。

QTEXTMODE = 0

QTEXTMODE = 1



REGENMODE	整数	図面	

オート再作図モードを制御します。
0 = オフ
1 = オン < 既定値 >

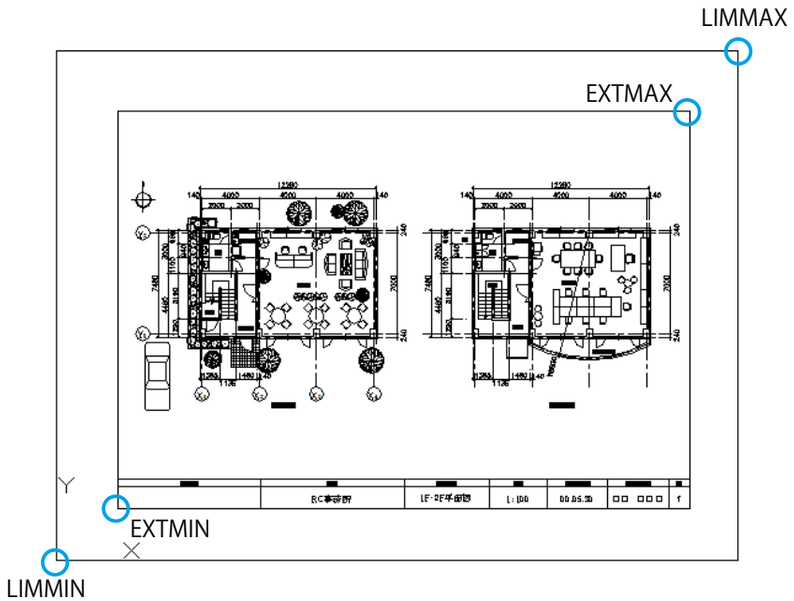
SNAPBASE	2 次元点	図面	

現在のビューポートでのスナップ原点を指定します。

SNAPUNIT	2 次元点	図面	

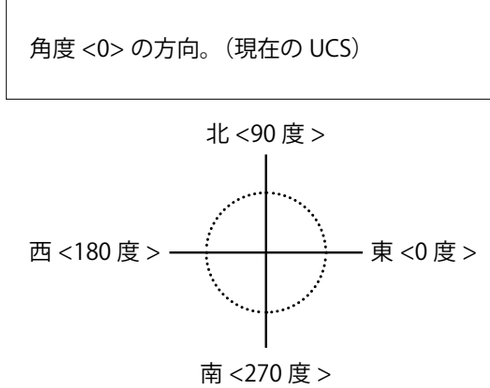
現在のビューポートに対する X と Y のスナップ間隔を指定します。

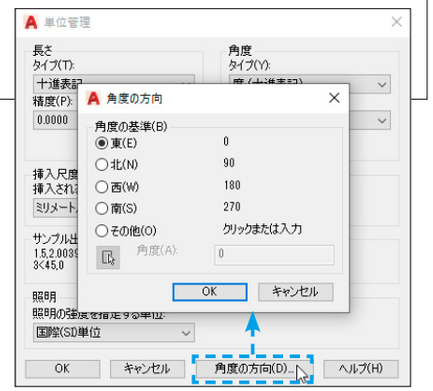
限界関係のシステム変数			
システム変数	種類	保管先	関連コマンド
EXTMAX	3 次元点	図面	
図形範囲の右上の頂点座標を表示します。新しく図形を描くと図面範囲は自動的に広がります。			
EXTMIN	3 次元点	図面	
図形範囲の左下の頂点座標を表示します。新しく図形を描くと図面範囲は自動的に広がります。			
LIMMAX	2 次元点	図面	LIMITS
ワールド座標で表示される現在の空間の図面範囲の右上頂点を設定します。			
LIMMIN	2 次元点	図面	LIMITS
ワールド座標で表示される現在の空間の図面範囲の左下頂点を設定します。			



図形範囲 (EXTMIN、EXTMAX) が、図面範囲 (LIMMIN、LIMMAX) を超えた場合、図面範囲も自動的に広がります。

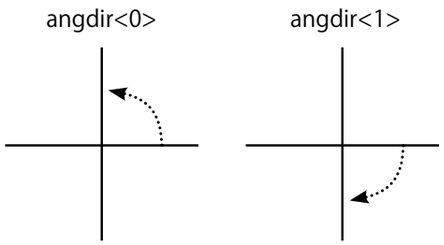
初期値関係のシステム変数			
システム変数	種類	保管先	関連コマンド
ANGBASE	実数	図面	
角度 <0> の方向。（現在の UCS）			

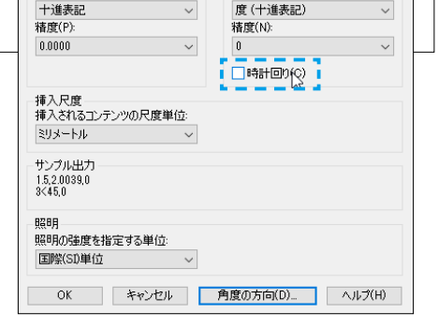




ANGDIR	整数	図面	UNITS

0 = 角度を反時計回りに計測します。< 既定値 >
1 = 角度を時計回りに計測します。

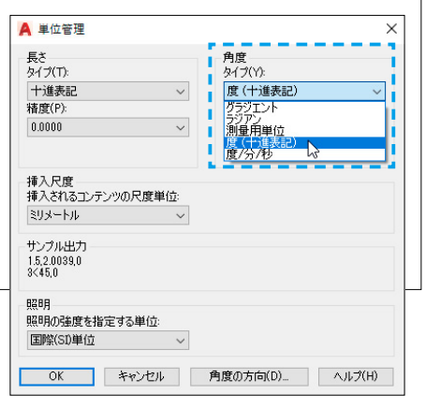




AUNITS	整数	図面	UNITS

角度の単位モードを制御します。

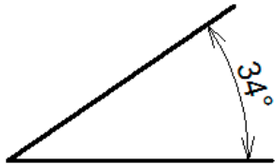
	例
0 = 十進数	35.00
1 = 度 / 分 / 秒	45d0'0"
2 = グラジェント	60.00g
3 = ラジアン	0.785398r
4 = 測量用単位	N 45d0'0" E

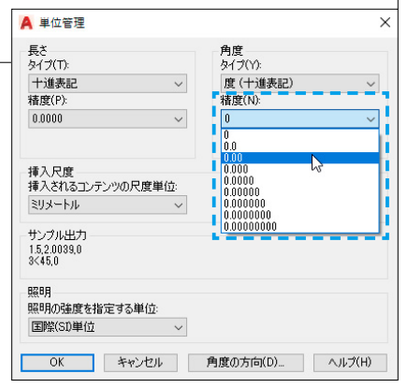


AUPREC	整数	図面	UNIT

角度の小数点以下の桁数を指定します。（0 ～ 8）

[AUPREC] を <2> に設定して、下図を [MEASUREGEOM] で角度を確認すると、角度 = 34.32 ° が表示されます。
しかし、寸法で角度を記入すると下図のように 34° と表示されます。角度寸法の設定は、[DIMADEC](寸法角度精度) で設定します。

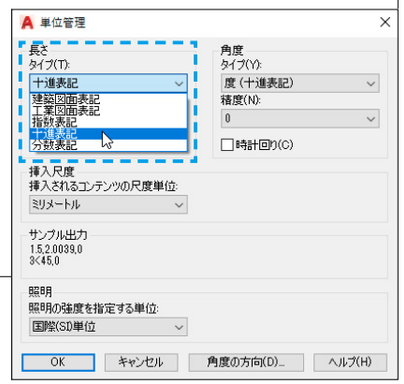




LUNITS	整数	図面	UNITS

線分の単位モードを設定します。

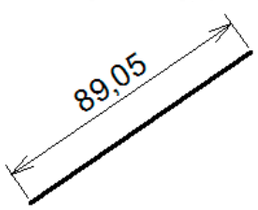
	例
1 = 指数表記	3.25E + 01
2 = 十進表記	25.80
3 = 工業図面表記	2' - 2.50"
4 = 建築図面表記	2' - 2 1/2"
5 = 分数表記	2 1/2

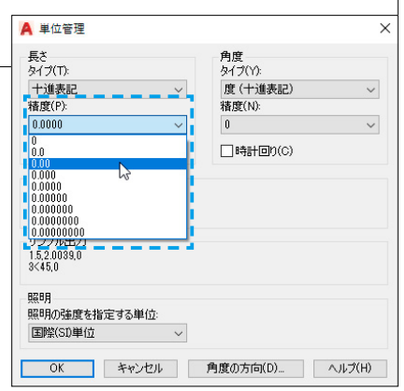


LUPREC	整数	図面	

長さの小数点以下の桁数を指定します。

[LUPREC] を <4> に設定して、下図を [MEASUREGEOM] で長さを確認すると、長さ = 89.0507 が表示されます。
しかし、寸法で長さを記入すると下図のように 89.05 と表示されます。長さ寸法の設定は、[DIMDEC](寸法精度) で設定します。





CHAMMODE	整数		CHAMFER

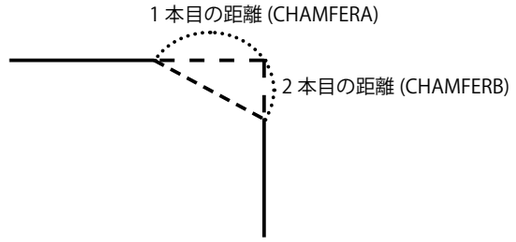
面取りを作成するときの入力方法を設定します。
0 = 2 つの面取り距離を指定します。< 既定値 >
1 = 面取り距離と角度を指定します。

CHAMFERA	実数	図面	CHAMFER

CHAMMODE が 0 に設定されているとき、面取りの 1 本目の距離を指定します。

CHAMFERB	実数	図面	CHAMFER

CHAMMODE が 0 に設定されているとき、面取りの 2 本目の距離を指定します。

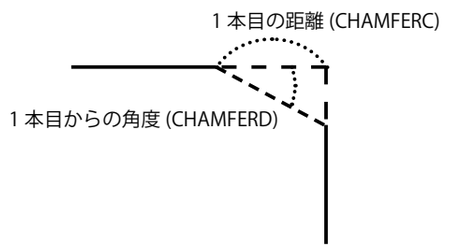


CHAMFERC	実数	図面	CHAMFER

CHAMMODE が 1 に設定されているとき、面取りの長さを設定します。

CHAMFERD	実数	図面	CHAMFER

CHAMMODE が 1 に設定されているとき、面取りの角度を設定します。



FILLETRAD	整数	図面	FILLET

FILLET コマンドで使用する円弧の半径を指定します。

情報関係のシステム変数			
システム変数	種類	保管先	関連コマンド

CECOLOR	文字列	図面	COLOR

現在の基本図形の色を指定します。< 既定値 "BYLAYER">

CELTYPE	文字列	図面	LINETYPE

現在の基本図形の線種を指定します。< 既定値 "BYLAYER">

CLAYER	文字列	図面	LAYER

現在の画層を指定します。

LASTPOINT	文字列		

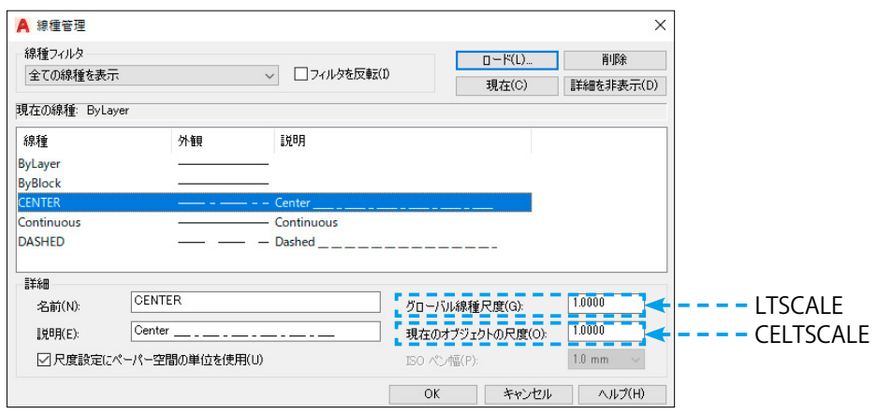
現在の空間の UCS 座標で表示される最後の入力点を指定します。キーボードから点入力の際に <@> で参照される点です。

LTSCALE	実数	図面	

図面全体の尺度を設定します。線種尺度はゼロを指定できません。

CELTSCALE	実数	図面	

現在のオブジェクトの線種尺度を設定します。新しいオブジェクトの線種尺度は、LTSCALE[線種　尺度] コマンドの設定に従います。

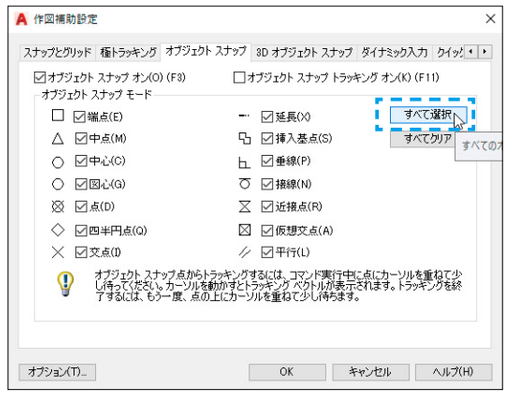


OSMODE	整数	図面	

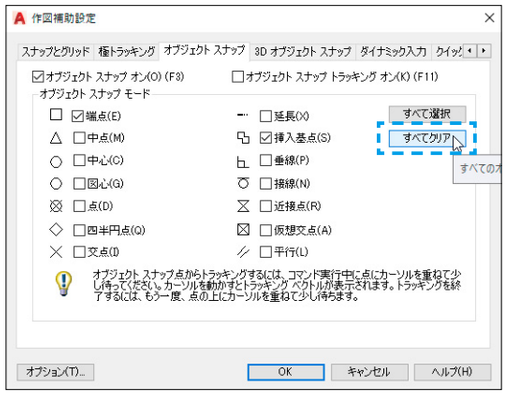
オブジェクト スナップのビット コードです。次の値の合計で指定します。

1 ＝ 端点	64 = 挿入基点	3 = 端点 + 中点
2 ＝ 中点	128 = 垂線	5 = 端点 + 中心
4 ＝ 中心	256 = 接線	7 = 端点 + 中点 + 中心
8 ＝ 点	512 = 近接点	9 = 端点 + 点
16 ＝ 四半円点	1024 = 図心	10 = 中点 + 点
32 ＝ 交点	2048 = 仮想交点	11 = 端点 + 中点 + 点
4096 = 延長	8192= 平行	16383 = すべて選択
		16384 = すべて解除

(setvar "OSMODE" 16383) → [すべて選択]



(setvar "OSMODE" 16384) → [すべて解除]

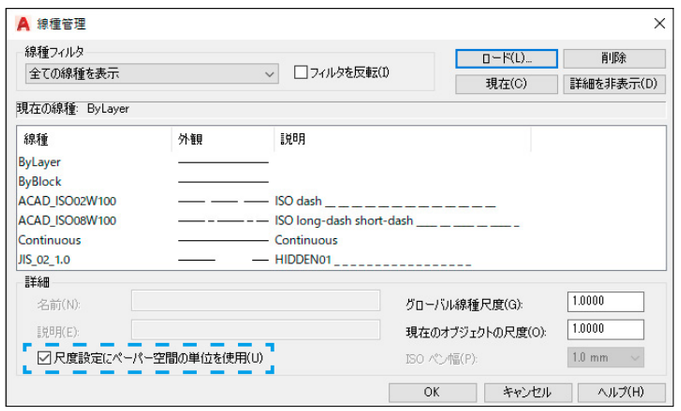


PSLTSCALE	整数	図面	

ペーパー空間の線の尺度を制御します。

0 = 線の尺度は、その図形がある空間の作図単位に従います。

1 = 線の尺度は、ペーパー空間の作図単位に従います。

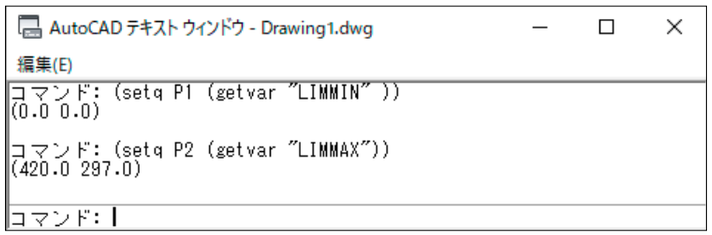


4 システム変数を取得する getvar 関数

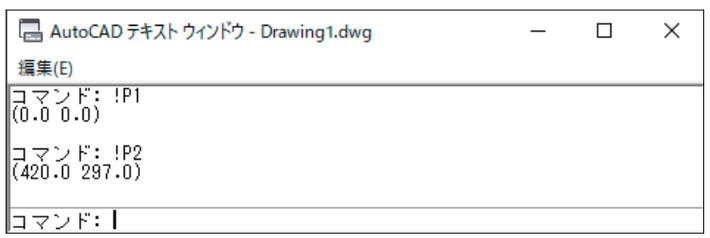
システム変数の値をコマンドラインから取得することができます。
このときに使用する関数が [getvar] です。

① getvar 関数を使用して、現在開いている図面の図面範囲を調べます。
新規に図面をオープンすると、初期値の図面範囲はミリ系で A3 の大きさになっています。
つまり、横 420 ×縦 297 の大きさです。
図面範囲のコマンドでは、最初に左下の座標を聞いてきて、次に右上の座標を聞いてきます。
このとき、左下の座標がシステム変数 [LIMMIN] にセットされ、右上の座標がシステム変数 [LIMMAX] にセットされます。
次のように記述します。

(setq P1 (getvar "LIMMIN")) ・・・ 図面範囲の左下の座標を変数 P1 に代入する。
→コマンドラインに、(0.0 0.0) が表示されます。
(setq P2 (getvar " LIMMAX")) ・・・ 図面範囲の右上の座標を変数 P2 に代入する。
→コマンドラインに、(420.0 297.0) が表示されます。



②キーボードから <!P1> と入力すると、コマンドラインに (0.0 0.0) が表示されます。
キーボードから <!P2> と入力すると、コマンドラインに (420.0 297.0) が表示されます。



Point!

システム変数には、次の 3 通りがあります。
①確認しかできない変数の例
・・[DISTANCE](読み込み専用)(保存されません)
②変更は出来るが保存はできない変数の例
・・[HIGHLIGHT](保存されません)
③変更も保存も可能な変数の例
・・[CHAMFERA](保存先は図面)

getvar 関数	LIMMIN と LIMMAX から図枠を作成する

システム変数 [LIMMIN] と [LIMMAX] から図面範囲の領域を示す四角形を作図します。

Step1 ー システム変数 [LIMMIN] から、図面範囲の最小座標（左下）を取得します。

Step2 ー システム変数 [LIMMAX] から、図面範囲の最大座標（右上）を取得します。

Step3 ー長方形 [rectang] コマンドを使い、左下を [LIMMIN]、右上を [LIMMAX] として作図します。

Step4 ー プログラムの最後に、<nil> を表示させないように (princ) を記述します。

```
(defun C:Box1( / P1 P2)
❶   (setq P1 (getvar "LIMMIN"))   ; 図面範囲の左下の座標      → Step1
❷   (setq P2 (getvar "LIMMAX"))   ; 図面範囲の右上の座標      → Step2
; 長方形コマンド [rectang] で長方形を作図する
❸   (command "RECTANG" P1 P2)     ; 長方形を作図             → Step3
❹   (princ)                                                  → Step4
) ;end
```

番号	関数名	説　明
①	getvar "LIMMIN"	システム変数 (LIMMIN) から図面範囲の最小座標（左下）を取得
②	getvar "LIMMAX"	システム変数 (LIMMAX) から図面範囲の最大座標（右上）を取得
③	RECTANG	長方形コマンド（２点を指示して作図）
④	princ	最後の行に、<nil> を表示させません

princ は、P1-141 を参照

5 システム変数に代入する setvar 関数

システム変数をコマンドラインからコントロールすることができます。
このときに使用する関数が [setvar] です。

① setvar 関数を使して、先ほどの図面範囲を変更します。
先ほどの図面の図面範囲はミリ系で A3 の大きさになっていますが、これをインチ系の A3 の大きさ、つまり、横 12 ×縦 9 の大きさに変更します。
左下の座標のシステム変数 [LIMMIN] に <0,0> をセットし、右上の座標のシステム変数 [LIMMAX] に <12,9> をセットします。

次のように記述します。

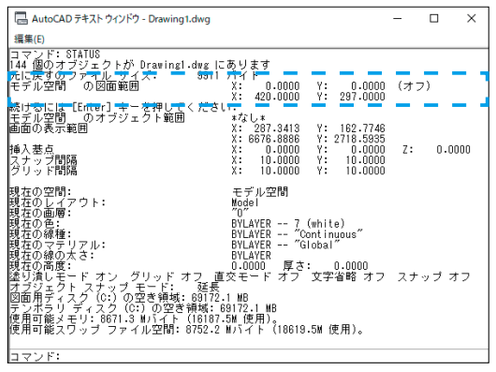
```
(setq  P1 (setvar “LIMMIN” (list 0 0 )))
→コマンドラインに、(0  0) が表示されます。
(setq P2  (setvar “ LIMMAX” (list 12 9)))
→コマンドラインに、(12  9) が表示されます。
```

②キーボードから <!P1> と入力すると、コマンドラインに (0.0 0.0) が表示されます。
キーボードから <!P2> と入力すると、コマンドラインに (12.0 9.0) が表示されます。

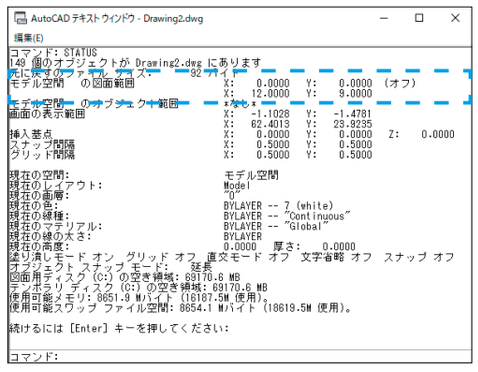
③ [設定] → [図面範囲] を選びます。
[左下のコーナー] に対して、<!P1> と入力します。
[右上のコーナー] に対して、<!P2> と入力します。

④ [ツール] → [情報] → [図面情報] で確認します。
左下の座標が (0,0)、右上の座標が (12,9) であることが分かります。(右図)

ミリ系の既定値

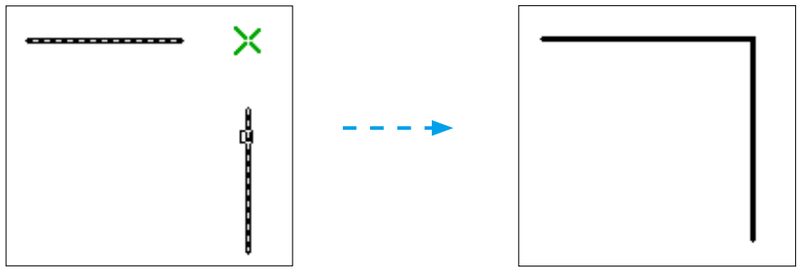


インチ系の既定値



getvar & setvar　フィレットの半径を <0> にした後、既定値に戻す

半径 <0> でフィレットします。終了後、フィレット半径を既定値に戻します。



Step1 ー 現在のシステム変数 [FILLETRAD] の値を変数 <fr> に代入します。（後で戻すため）

Step2 ー システム変数 [FILLETRAD] の値を <0> にセットします。

Step3 ーフィレット [fillet] コマンドを使い、2 本の線分をフィレットします。
（2 本の線分は、ユーザーが指示します。<pause> を 2 回使います。）

Step4 ー システム変数 [FILLETRAD] の値を初期値に戻します。

Step5 ー プログラムの最後に、<nil> を表示させないように (princ) を記述します。

```
(defun C:Fillet0( / fr)
❶   (setq fr  (getvar  "filletrad"))  ; フィレット半径を得る      → Step1
❷   (setvar  "filletrad"  0)  ; フィレット半径を <0> にセット     → Step2
❸   (command  "fillet"  pause  pause)  ; フィレットを実施         → Step3
❹   (setvar  "filletrad"  fr)  ; フィレット半径を元に戻す         → Step4
❺   (princ)                                                      → Step5
) ;end
```

番号	関数名	説　明
①	getvar "filletrad"	システム変数 [filletrad] の値を取得（最後に戻すため）
②	"filletrad" 0	フィレットの半径を <0> にセット
③	pause pause	pause は入力待ち（ユーザーが 2 カ所指示するのを待ちます）
④	setvar "filletrad"	システム変数 [filletrad] の値を元に戻します
⑤	princ	最後の行に、<nil> を表示させません

pause は、P1-42 を参照

6 定義済み変数（定数）

次に示す定義済み変数（定数）は、AutoLISP でよく使用します。

AutoCAD の定義済み変数（定数）	
定数	説　明
PAUSE	2 つの円記号から成る文字列 (¥¥) として定義されています。この変数は、ユーザー入力待ち (一時停止) を行うために command 関数で使用します。
PI	定数 π (パイ) として定義されています。これは、約 3.14159 に評価されます。
T	定数 T として定義されています。これは、nil でない値として使用します。
nil	「何も無い」という意味です。[値が無い]、[ファイルが無い]、[リストが無い] など。また、IF 関数において真 (T) でない時の偽 (nil) としても使われます。

これらの変数の値は、setq 関数で変更できます。しかし、他のアプリケーションがこれらの値が一定であることを前提にしている可能性がありますから、これらの変数は変更すべきではありません。

PAUSE	command 関数の実行中に、ユーザーの選択待ちにします。

LINE（線分）コマンドで表示されるプロンプトは、以下の通りです。

```
LINE 1 点目を指定 :                    → マウスで 1 点目を指示します。(pt1)
次の点を指定 または [ 元に戻す (U)]:    → マウスで 2 点目を指示します。(pt2)
```

指示された pt1 点と pt2 点に線分を作図します。
これを LISP で記述した場合は、(command "LINE" pt1 pt2 "") となります。

もし、"LINE" の後で pt1 と pt2 をユーザーに画面上から指示させるためには、[pause] 関数を使って以下のように記述します。

```
(command "LINE" pause pause "")        → 最後の <""> は空打ちです。
```

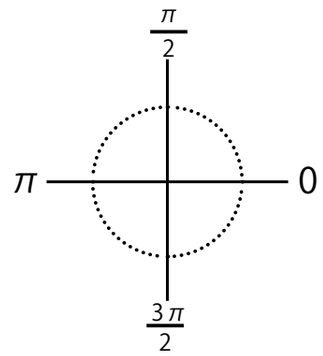
pause 関数を使用すると、pt1 と pt2 が事前に判っていなくても、"LINE" コマンドが使用出来ます。

このように、画面上で指示を待つタイプのコマンドで使用出来ます。
次の例は、移動コマンドの例です。

```
(command "MOVE" pause "" pause pause)

オブジェクトを選択 :                   → 最初の pause（"" は選択終了のエンター）
基点を指定 または [ 移動距離 (D)]      → 二番目の pause
< 移動距離 >: 目的点を指定             → 三番目の pause
```


pi	円周率のπですが、キーボードにはこの記号が無いので、pi で代用。

<pi> は円周率ですから、LISP の関数では angle 関数や getangle 関数、getorient 関数で使用されます。



0 度は <0> ラジアン
90 度は < 約 1.57079> ラジアン
180 度は < 約 3.14159> ラジアン
270 度は < 約 4.71238> ラジアン

T	nil でない値として使用します。（式が正しい）（リストが空でない）

変数 <a> と <b> の値が等しいかどうかをチェックして、等しければ（T）A の式を行い、等しくなければ（nil）Bの式を行う LISP は以下のようになります。

```
(If (= a b)                    → a と b を比較します。
    (setq c (+ a b))           → 等しければ (T)、a と b を加算します。
    (setq c (-  a b))          → 等しくなければ (nil)、a から b を減算します。
)
```

nil	IF 関数において真 (T) でない時やリストが空であるとき。

値が代入されていない AutoLISP の変数を、nil といいます。これは、空白とは異なり (空白は文字とみなされます)、また 0(ゼロ) とも異なります (0 は数値とみなされます)。
そのため、変数の現在値をチェックするほかに、変数に値が代入されているかどうかを調べるためにテストできます。

各変数は少量のメモリを消費するため、値が必要としなくなったときには、その変数名を再び使用するか、またはその変数に nil を代入します。変数に nil を代入すると、その変数の値を格納するために使用されていたメモリが解放されます。変数 < pt1> が不要になったときは、次の式を使用してその変数の値をメモリから解放できます。

```
(setq pt1 nil)
!pt1                           → キーボードから、!pt1 と入力します。
nil                            → nil が返ります。
```

7 AutoLISP の変数のまとめ

① AutoCAD の定数（定義済み変数）

AutoCAD の関数（Line や Circle）と同様に、これらの定義された変数はユーザー側で変更できません。

① pause ・・ユーザーの選択待ちの関数です。
② pi ・・円周率です。ユーザー側で (setq pi 10) とすると、そのセッションでは円周率は <10> になってしまいます。
③ T ・・真偽判定などに使われる重要な定数です。
④ nil ・・上記同様の重要な定数です。

② システム変数

システム変数には、動作環境に関する設定値や AutoCAD のコマンドの設定値が格納されています。
現在開いている図面のシステム変数を確認するには、キーボードから (getvar " システム変数名 ") と入力します。
またシステム変数を変更するには、(setvar " システム変数名 " 値) と入力します。
ただ、システム変数には確認するだけで変更できない変数もあります。

③ ユーザーが指定できる変数

①グローバル変数・・ユーザー関数 (defun 関数）が終了した後でも、メモリーに残ります。
何も宣言しなければグローバル変数になります。
関数内で宣言すれば、ローカル変数になります。
②ローカル変数 ・・その関数内だけで有効です。関数が終了すると <nil> にセットされます。

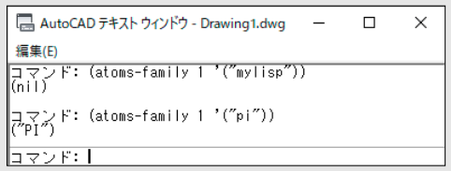
グローバル関数とローカル関数の関係は次の通りです。

```
(defun C:test( a / b)
      (setq c (+ a b))
)
```

変数 [a] と [b] はローカル変数、[c] はグローバル変数です。
このうち、[a] は引数とも呼ばれます。
役割としては、[a] が [受け取るローカル変数]、[b] が [与えるローカル変数] になります。

Point!

ユーザーが指定する変数名（シンボル名）が、すでに定義済みかどうか調べるには
キーボードから、(atoms-family 1 '(" シンボル名 ")) と入力します。
定義されていなければ、nil が返ります。



左図では、"mylisp" で調べると (nil) が返ります。
しかし、"pi" で調べると ("PI") が返りましたので、"pi" は AutoCAD のシンボルとしてすでに使われていることが判ります。

第 2 章 AutoLISP の関数

AutoLISP の中心となる関数のグループです。
リストの処理方法が他のプログラムとは特徴的ですが、
CAD データを扱うには最も適した関数と言えます。

この章では以下のことを学びます。

第 1 節　リスト操作関数

円を作図し、その図形情報を確認すると、以下のように <( )> が連結して表示されます。
これら 1 つ 1 つの <( )> が連結されたものをリストと呼びます。
((-1 . < 図形名 : 7ef03550>) (0 . "CIRCLE") (330 . < 図形名 : 7ef01cf8>) (5 . "222") (100 . "AcDbEntity")
(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (101066.45 1562.17 0.0) (40 . 97.2096) (210 0.0 0.0 0))

LISP は LISt Processor という名前の通り、リストは LISP における基本的なデータ構造です。
この節では、要素をリストに加えたり、削除したり、入れ替えたり、新たに組み直したりする関数を学びます。

使用頻度の高い AutoLISP のリスト操作関数を、次の表に示します。（順不同）

①リストから要素を取り出す関数	
関　数	説　明
(car lst)	リストの先頭の要素を返します。
(cadr lst)	リストの 2 番目の要素を返します。
(cddr lst)	リストの 3 番目の要素を返します。
(cdr lst)	指定されたリストの先頭の要素以外のすべての要素を含むリストを返します。
(last lst)	リストの最後の要素を返します。
(nth n lst)	リストの n 番目の要素を返します。（n は 0 から始まります。）
(assoc item alist)	連想リストの要素を検索し、指定された要素が含まれる連想リスト項目を返します。

②リストに要素を加える関数	
関　数	説　明
(list [expr ...])	任意の数の式を受け取り、それらを 1 つのリストに結合します。
(cons new-first-element lst)	リストの先頭に要素を追加します。また、ドットリストを作成します。
(append lst ...)	任意の数のリストを受け取り、それらを 1 つのリストにまとめます。
(apply 'function list)	指定された関数に引数のリストを渡し、その関数を実行します。
(foreach name lst [expr ...])	リストの個々の要素をリストに渡します。
(mapcar function list1 ... listn)	関数にリストの各要素を引数として渡します。

③リストの要素を入れ替える関数	
関　数	説　明
(subst newitem olditem lst)	リストで古い項目を検索し、古い項目があるたびに新しい項目で置き換えたリストを返します。
(reverse lst)	要素の順番を反転させたリストを返します。
(member expr lst)	指定された式がリスト内に存在するかどうかを検索し、初めに検出した式以降のリストを返します。（サブリストを作成します。）
(acad_strlsort)	文字列のリストを文字コード順にソートします。

④リストを調べる関数	
関　数	説　明
(length lst)	リストの要素数を表す整数を返します。
(listp item)	指定された項目がリストかどうかを調べます。
(tblsearch table-name symbol [setnext])	シンボル テーブル内のシンボル名を検索します。
(tblnext table-name [rewind])	シンボル テーブル内の次の項目を取得します。

1　リストから要素を取り出す関数

CAR　リストの中の指定した 1 番目の要素を返します

(car (list 10 20 30 40))　→ 10

CADR　リストの中の指定した 2 番目の要素を返します

(cadr (list 10 20 30 40))　→ 20

CDDR　リストの中の指定した 3 番目の要素を返します

(cddr (list 10 20 30 40))　→ 30

CDR　リストの先頭の要素以外のすべての要素を含むリストを返します

(cdr (list 10 20 30 40))　→ (20 30 40)

LAST　指定したリストの最後の要素を返します

(last (list 10 20 30 40))　→ 40

NTH　リストの中の指定した番目の要素を返します

(nth 2 (list 10 20 30 40))　→ 30（番号は <0> から始まります。）

この関数には、2 つの引数を指定します。最初の引数は、どの項目を返すかを指定する整数です。最初の引数として 0(ゼロ) を指定するとリストの最初の項目が返され、1 を指定すると 2 番目の項目が返されます。2 番目の引数には、項目を取り出すリストを指定します。

(setq a (list 1 2 3 4 5))
(setq b (nth 3 a))
!b　→ 4

ASSOC　連想リストの中から、指定された項目を取り出します

(setq pt '((a 10)(b 20)(c 30)))　→ ((A 10) (B 20) (C 30))
(assoc 'a pt)　→ (A 10)
(assoc 'd pt)　→ nil（指定された項目が無い場合）
(cdr (assoc 'a pt))　→ (10)

<car><cadr><cddr>	リストの要素を取り出す

List 関数を使うと、複数の数値を組み合わせて、X,Y,Z の座標を構成することができます。
その反対に、すでに組み合わされている座標から、X の数値、Y の数値、Z の数値だけを取り出す関数が <car>、<cadr>、<cddr> です。

①まず、X の座標 10.0　Y の座標 20.0　Z の座標 0.0　のリストを作ってみましょう。
(setq P1 (list 10.0 20.0 0.0)) となります。

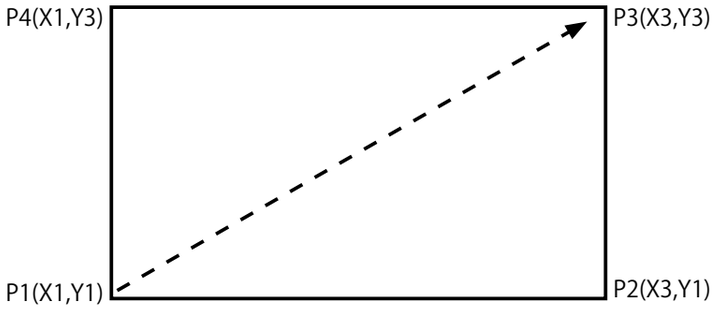
②このリストから、X 座標を取り出す関数が <car> です。X 座標の変数を Px とすると
(setq Px (car P1)) となります。

③同様に、Y 座標を取り出す関数は <cadr> です。Y 座標の変数を Py とすると
(setq Py (cadr P1)) となります。

④同様に、Z 座標を取り出す関数は <cddr> です。Z 座標の変数を Pz とすると
(setq Pz (cddr P1)) となります。

下図の四角形を作図するプログラムで考えてみましょう。

マウスで点 P1(X1,Y1) と点 P3(X3,Y3) を指示することによって、他の 2 点 (P2 と P4) を求めます。



点 P2 の X 座標は、点 P3 の X 座標と同じです。また、Y 座標は点 P1 の Y 座標と同じです。
これより、点 P2 の座標は、点 P3 の X 座標と点 P1 の Y 座標を組み合わせたものになります。

①点 P3 の X 座標を取り出す。　X3 = (car P3)
②点 P1 の Y 座標を取り出す。　Y1 = (cadr P1)
③上記 2 つの数値を list 関数で 1 つにして、点 P2 の座標にする。
(setq P2 (list (car P3) (cadr P1)))
④同様にして、点 P4 の座標は　(setq P4 (list (car P1) (cadr P3))) になります。

この四角形を作図する関数名を Box2.lsp として、作成してみましょう。

```
❶ (defun C:Box2( / p1 p2 p3 p4)   ; 関数の宣言
❷   (setq p1 (getpoint "¥n 一点目を指示 : " ))   ;p1 の座標を取得
❸   (setq p3 (getpoint p1 "¥n 二点目を指示 : " ))   ;p3 の座標を取得
   ;p1 と p3 を組み合わせて、p2 と p4 の座標を作成
❹   (setq p2 (list (car p3) (cadr p1)))   ;p2 の座標
❺   (setq p4 (list (car p1) (cadr p3)))   ;p4 の座標

❻   (command "LINE" p1 p2 p3 p4 "C" )   ; 線分コマンドで作成
) ;end
```

getpoint は、P1-52 を参考

list は、P1-50 を参考

番号	関数名	説　明
①	defun	関数の宣言になります。defun C: プログラム名 ( )　は決まりです。 (/ p1 p2 p3 p4) は、この中だけで使用する変数名です。これを省いた場合はメモリ上に残りますので、他の関数でも使用できます。この場合は、このプログラムが終了するとメモリから開放されます。
②	getpoint	プロンプトに表示されるメッセージを見て、ユーザーが指示した点の座標を変数 p1 にセットします。
③	getpoint p1	ユーザーがマウスで 2 点目を指示した点の座標を変数 p2 にセットします。 getpoint p1 とすると、p1 からラバーバンドが表示されます。
④	setq p2	点 p3 の X 座標と点 p1 の Y 座標から、点 p2 の座標を作ります。
⑤	setq p4	点 p1 の X 座標と点 p3 の Y 座標から、点 p4 の座標を作ります。
⑥	command	4 つの点が取得できたので、線分コマンドで四角形を作図します。 最後の " C" は最初の点 (P1) まで結ぶ (Close) 意味です。

最後の括弧) は忘れてはいけません。
AutoLISP はこのように、始めの括弧と終わりの括弧の中に記述していきます。
括弧の数が合わないときは、エラーになります。

メモリにある数値には " と " で囲むことはいりませんが、ユーザーがキーボードから入力しなければならないものは、" と " で囲みます。この場合は、<LINE> と <C> です。

Point!

car、cadr、cddr と nth との関係

(setq a (list 10 20 30)) であるとき、
X 座標は、(car a) = (nth a 0)　→ x = 10
Y 座標は、(cadr a) = (nth a 1)　→ y = 20
Z 座標は、(cddr a) = (nth a 2)　→ z = 30

2 リストに要素を加える関数

list 関数	任意の数の式を受け取り、それらを 1 つのリストに結合します

複数の要素を 1 つのリストにします。
変数 [pt] に複数の要素 <10.0 20.0 30.0> をセットするには、
→ (setq pt (list 10.0 20.0 30.0)) または、(setq pt '(10.0 20.0 30.0)) とします。

CONS	リストの先頭に指定した要素を付加したリストを作成します

(cons '10 '(20 30 40)) → (10 20 30 40)
(cons '(10 20) '(30 40)) → ((10 20) 30 40)
下記のように、リストではなくアトムの場合は、ドット・ペアを返します。☞ドット・ペアは、P1-55 を参考
(cons 'a 10) → (A . 10)

APPEND	複数のリストを 1 つにまとめます

append 関数は、任意の数のリストを 1 つのリストにまとめます。
したがって、この関数の引数は、すべてリストになります。
(append (list 10 20) (list 30 40)) → (10 20 30 40)
(setq a '(30 40)) (append (list 10 20) a) → (10 20 30 40)

Point!

既存のリストを変更する関数は 3 つあります。
append 関数は、新しい項目を末尾に追加したリストを返します。
cons 関数は、新しい項目を先頭に追加したリストを返します。
subst 関数は、古い項目があるたびにそれを新しい項目で置き換えたリストを返します。

これらの関数は元のリストを変更せず、変更された新しいリストを返します。
従って元のリストを変更するには、明示的に古いリストを新しいリストで置き換えることになります。

FOREACH	リストの全ての要素に対して式を評価します

書式：(foreach 変数 リスト 評価する式) → 評価する式が無いときは、nil を返します。
コマンド : (foreach a '(1 2 3) (1+ a)) → 4（最後の値 <1+3> が返る）

(foreach a '(a b c)(print a)) → (print a)(print b)(print c) と同じ
A
B
C C → foreach 関数は、リスト内の各要素を表示して、最後の要素 <C> を返します。

MAPCAR	指定された各要素を引数として、関数を実行した結果を返します

(mapcar '1+ '(1 2 3)) → (2 3 4)
最初の引数で与えられた関数を、2 番目の引数 (リスト) 内の連続する要素に適用します。
(1+ 1) -> 2 、(1+ 2) -> 3 、(1+ 3) -> 4

(mapcar 'setvar '("cmdecho" "osmode") '(0 1)) → (setvar "cmdecho" 0) (setvar "osmode" 1) と同じ

APPLY	指定された関数に引数のリストを渡し、その関数を実行します

(apply '+ '(1 2 3)) → 6
(apply 'strcat '("Auto" "LISP")) → "AutoLISP"

Point!

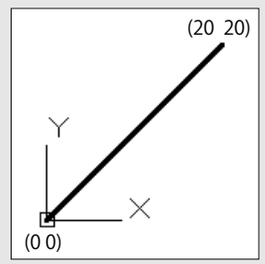
下記の①②③は同じ線分 (<0,0> から <20,20>) を作図します。

変数 [pt_lst] に (0,0) と (20,20) と ("") のリストをセットします。
(setq pt_lst '((0 0) (20 20) (""))) → ((0 0) (20 20) (""))

① (command "line")(foreach a pt_lst (command a))

② (command "line")(mapcar 'command pt_lst)

③ (command "line")(apply 'command pt_lst)



<getpoint><list>	点（座標）リストを作成する

マウスで座標リストを作成する　・・・　getpoint

①マウスで画面上の位置を指示したときに座標が取得できます。
(setq pt (getpoint "¥n 位置を指定：")) と入力します。
→コマンド : (setq pt (getpoint "¥n 位置を指定："))
マウスで適当な位置を指示します。
→位置を指定：(756.301 488.818 0.0)

( ) の中の数値が、XYZ の座標値です。画面で指示する位置は平面ですが、Z 座標値も入力されています。これは高度（ELEVATION）の初期値が <0.0000> に設定されていますので、Z 座標には自動的に高度設定の数値が入ります。

②システム変数 <ELEVATION> には、現在の空間の現在のビューポートでの現在の UCS を基準にした現在の高度が格納されています。
この高度を変更するには、setvar 関数を使います。
(setvar "ELEVATION" 100.0)　→ マウスで画面を指示すると、Z 座標には自動的に <100.0> がセットされます。
①と同様に画面上で適当な位置を指示してみます。
コマンド : (setq pt (getpoint "¥n 位置を指定："))
位置を指定：(901.028 731.872 100.0)　→ Z 座標に <100.0> がセットされています。

高度（ELEVATION）を元の位置 <Z = 0.0> に戻すには、再度 setvar 関数を使います。
(setvar "ELEVATION" 0.0)

キーボードから座標リストを作成する　・・・　list

①キーボードから XYZ の座標を作成するには、list 関数を使います。
(setq pt (list 100.0 200.0))　→ X 座標に <100.0>、Y 座標に <200.0> をセット
この時の Z 座標は、その時点のシステム変数 <ELEVATION> の値が自動的にセットされます。
ですから、<ELEVATION> の値が <0.0> のときは、実際の座標値は (100.0 200.0 0.0) になります。

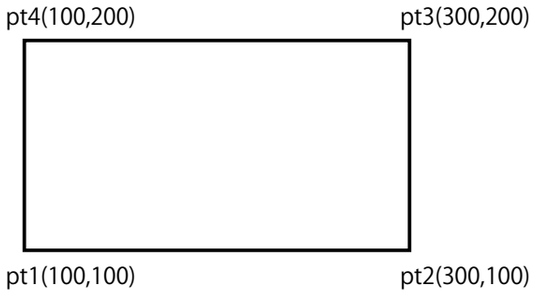
②現在の <ELEVATION> ではない Z 座標値であれば、list の中に明示的に記述します。
(setq pt (list 100.0 200.0 300.0))　→ Z 座標に <300.0> がセットされますが、システム変数 <ELEVATION> は <0.0> のままです。

③リストに変数を使用することもできます。
(setq pt1 10.0 pt2 20.0 pt3 30.0)
(setq pt (list pt1 pt2 pt3))　→ pt には (10.0 20.0 30.0) がセットされます。

④ list 関数の代わりに、quote 関数を使用してリストを作成できます。
quote 関数は、次に示すように式を評価せずに返します。
(setq p1 (quote (10.0 20.0 30.0)))　→ pt には (10.0 20.0 30.0) がセットされます。

⑤ quote 関数の代わりに点の (') を使用することもできます。
以下のコードは、上のコードと同じ結果になります。
setq p1 '(10.0 20.0 30.0))　→ pt には (10.0 20.0 30.0) がセットされます。

キーボードから座標を入力して四角形を作図する



Step1 ー各コーナーの座標をセットします。
(setq pt1 (list 100.0 100.0))
(setq pt2 (list 300.0 100.0))
(setq pt3 (list 300.0 200.0))
(setq pt4 (list 100.0 200.0))

Step2 ー線分コマンドで作図する場合。
(command "LINE" pt1 pt2 pt3 pt4 "C")　→ "C" は Close の C です。

Step3 ー長方形コマンドで作図する場合。
(command "RECTANG" pt1 pt3)

3 リストの要素を入れ替える関数

SUBST	リストの中の古いデータを新しいデータと入れ替えます

```
(setq  en1  (list  10  20  30  40))
(setq  en1  (subst  50  20  en1))
```

コマンド : !en1　　→ コマンドラインに <!en1> と入力します。
(10 50 30 40)　　→ 要素 <20> が <50> と入れ替わっています。

古いリストに同じデータが複数ある場合は、すべて新しいデータと置き換わります。

```
(setq  en1  (list  10  30  30  40))
(setq  en1  (subst  50  30  en1))
```

コマンド : !en1　　→ コマンドラインに <!en1> と入力します。
(10 50 50 40)　　→ 2 つの要素 <30> が <50> と入れ替わっています。

REVERSE	リストの要素の順番を反転させたリストを返します

```
(reverse  (list  10  20  30  40))          → (40 30 20 10)
(reverse  (list  '(100,100)  '(200,200)))  → ((200,200) (100,100))
```

MEMBER	指定された式がリスト内に存在するかどうかを検索します

```
(member  '20 '(10 20 30))                  → (20 30)
(member  '(30 30) '((10 10)(20 20)(30 30)))  → ((30 30))
```

リストが見つかると、それ以降のリストを返します。

ACAD_STRLSORT	文字列のリストを文字コード順にソートします

```
(setq  list_x (list  "Abc"  "123"  "Acb"  "231"))
(acad_strlsort  list_x)                    → ("123" "231" "Abc" "Acb")
```


特殊なリスト（ドット・ペア）の作成
特殊なリスト (ドット・ペア) とは。

点の座標のリストは、(X Y Z) の組み合わせで表示されます。(100.0 200.0 300.0) である場合は、X 座標が <100.0>、Y 座標が <200.0>、Z 座標が <300.0> であることが判ります。

一方、画面上のオブジェクト（例えば線分）は、画層や色、線種などの情報を持っています。

線分 ―――――――――

①上図のような線分を作成して、キーボードから以下のように入力します。
(setq ent1 (entlast))　　→（例）< 図形名 : 7ef03990> が返ってきます。
関数 entlast は、最後に作成したオブジェクトを変数にセットします。
この図形の中身を調べる関数が、entget です。

② (setq ent2 (entget ent1)) と入力すると、このオブジェクトの情報が表示されます。
((-1 . < 図形名 : 7ef03990>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A2") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 605.415 418.055 0.0) (11 1137.5 418.055 0.0) (210 0.0 0.0 1.0))
上のリストは、( ) が連なっていますが、この 1 つ 1 つの ( ) がドット・ペアと呼ばれます。
このリストから判るのは、オブジェクトの種類 <(0 . "LINE")>、画層 <(8 . "0")>、
始点座標 <(10 605.415 418.055 0.0)>、終点座標 <(11 1137.5 418.055 0.0)> 等です。

③このリストには、必ず 2 つの要素が格納されています。この 2 つの要素はピリオド (.) で区切られます。 2 つの要素の内、最初の要素をグループコードと呼びます。
上記の中の (8 . "0") では、グルー　プコードの <8> は画層を意味します。
その次の <"0"> は線分の画層の名前を意味します。
上記の線分のグループコードに関する主なものを示します。

線分のグループ コード	
グループコード	意　味
0	オブジェクトタイプ ("LINE")
8	画層名
10	始点 (WCS)　DXF: X 値、APP: 3D 点
11	終点 (WCS)　DXF: X 値、APP: 3D 点
210	押し出し方向 (省略可能、既定 = 0, 0, 1)　DXF: X 値、APP: 3D ベクトル

entlast は、P1-209 を参考
entget は、P1-218 を参考

assoc 関数　特殊なリスト（ドット・ペア）を取り出す

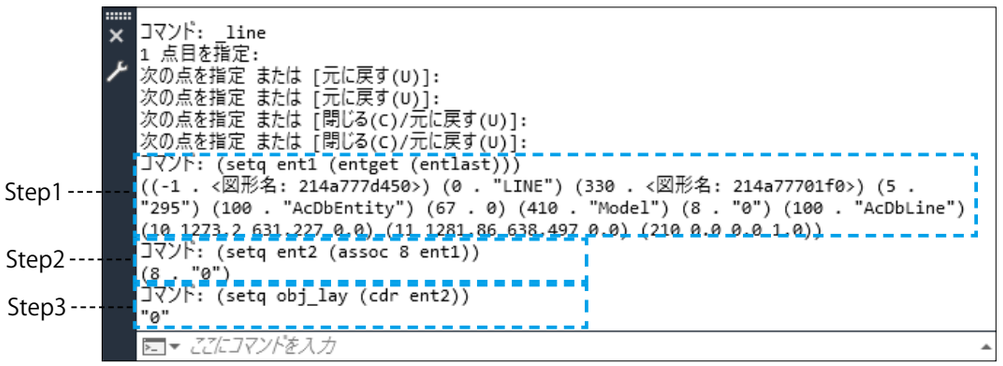
2 つの要素からなるリスト（連想リスト）のグループコードを検索して、一致する 1 組の要素を取り出すことができます。
書式は、(assoc　グループコード　ドット・ペア < 連想リスト >) です。

図面に最後に追加された図形を調べ、その画層名を取得するには次のようにします。

Step1　(setq ent1 (entget (entlast)))　→ 最後の図形のエンティティ データを取り出します。
　　コマンド : !ent1　→ コマンドラインに <!ent1> と入力します。
　　((-1 . < 図形名 : 214a777d450>) (0 . "LINE") (330 . < 図形名 : 214a77701f0>) (5 . "295")
　　(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine")
　　(10 1273.2 631.227 0.0) (11 1281.86 638.497 0.0) (210 0.0 0.0 1.0))

Step2　(setq ent2 (assoc 8 ent1))　→ グループ コード <8> のリストを取り出します。
　　コマンド : !ent2　→ コマンドラインに <!ent2> と入力します。
　　(8 . "0")　→ グループ コード <8> のドット・ペアが表示されます。

Step3　(setq obj_lay (cdr ent2))　→ リストの 2 番目以降を取り出します。
　　コマンド : !obj_lay　→ コマンドラインに <!obj_lay> と入力します。
　　"0"　→ 画層名の <"0"> が表示されます。



cons 関数　特殊なリスト (ドット・ペア) を作成する

①画層名が <"1"> のドット・ペアを作成してみましょう。
　要素を組み合わせて、ドット・ペアを作成する関数は cons です。
　以下のように記述します。
　　(cons　8 “1”)　となります。
　cons 関数には 2 つの引数が必要です。第 1 の引数はグループコード、第 2 の引数はそのコードの情報です。（画層のグループコードは <8> です。）

②上記のように入力して、この情報を変数 ent1 にセットします。
　　(setq ent1 (cons　8 “1”))

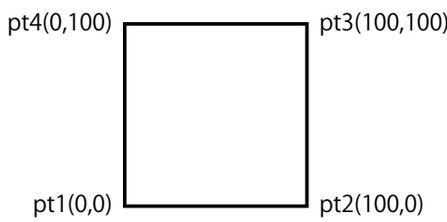
③キーボードから、!ent1 と入力します。次のドット・ペアが返されます。
　　(8. “1”)

> cons 関数に変数も使用出来ます。
> 変数 pt に X の座標値 <100.0>、Y の座標値 <100.0> をセットします。
> 　(setq pt (list 100.0　100.0))
> コマンドラインから次のように入力します。
> 　コマンド : (cons 10 pt)
> 　→ (10 100.0 100.0) が返ります。グループコード <10> は線分の始点の座標です。

entmake 関数で、四角形を作図する

entmake は、P1-222 を参考

entmake 関数は、entget 関数の反対の役割をします。従って entmake 関数の引数は、entget 関数が返すのと同じ形式の図形定義データになります。下図の四角形を entmake 関数で作成します。



グループコード <10> は線分の始点の座標
グループコード <11> は線分の終点の座標

Step1 ー線分 (pt1-pt2) の作図
　(entmake (list (cons 0　"line") '(10　0.0　0.0 0.0) '(11　100.0　0.0　0.0)))
Step2 ー線分 (pt2-pt3) の作図
　(entmake (list (cons 0　"line") '(10　100.0　0.0 0.0) '(11　100.0　100.0　0.0)))
Step3 ー線分 (pt3-pt4) の作図
　(entmake (list (cons 0　"line") '(10　100.0　100.0 0.0) '(11　0.0　100.0　0.0)))
Step4 ー線分 (pt4-pt1) の作図
　(entmake (list (cons 0　"line") '(10　0.0　100.0 0.0) '(11　0.0　0.0　0.0)))

subst、entmod	特殊なリスト（ドット・ペア）を入れ替える

図形リスト内のドット・ペア（連想リスト）を入れ替えるには、subst 関数を使います。
これは、図形リストの古いドット・ペアと新しいドット・ペアとを入れ替えます。

書式は、(subst　新しいドット・ペア　古いドット・ペア　図形リスト) です。
P1-50 の図形 ent1 のリストは次のようになっています。

((-1 . < 図形名 : 7ef03550>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "222") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 1403.87 1390.95 0.0) (11 1816.94 1540.2 0.0) (210 0.0 0.0 1.0))

この図形 ent1 の画層名は "0" ですが、"1" に変更するには、以下のようにします。
①画層が "1" のドット・ペア（連想リスト）を事前に作成します。

(setq new_lay (cons 8 "1")) → 新しいドット・ペア (8 . "1") を作成して変数 new_lay に代入します。

②最後に作図した図形を変数 ent1 にセットします。

(setq ent1 (entlast)) → 図形名が変数 ent1 にセットされます。

③図形 ent1 のリストを取得します。取得する関数は、entget です。

(setq old_obj (entget ent1)) → 最後に作成した図形 (entlast) の図形リストを変数 old_obj にセットします。

④図形リスト old_obj から画層のドット・ペア（連想リスト）を取得します。

(setq old_lay (assoc 8 old_obj)) → 図形リスト old_obj の画層のドット・ペアを変数 old_lay に代入します。

⑤図形リスト old_obj の画層のドット・ペア（連想リスト）を new_lay と入れ替えます。

(setq new_obj (subst new_lay old_lay old_obj))

⑥ subst 関数で入れ替えた後は、必ず entmod 関数でデータの更新処理を行います。

(entmod new_obj)
これで ent1 の画層は、"0" から "1" に変更されます。

(連想リスト) "Association List" の訳語ですが、[連結リスト] とか [結合リスト] の方が、判りやすいかと思います。

4 リストを調べる関数

LENGTH	指定したリストの要素数を返します

(length (list 10 20 30 40)) → 4

最後に作図したオブジェクトを取得して、en1 にセット → (setq en1 (entlast))
この図形 <en1> の要素数を得るには、length 関数で確認できます。
コマンド : !en1 → コマンドラインに < !en1 > と入力。
((-1 . < 図形名 : 7ef03550>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "222") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 1403.87 1390.95 0.0) (11 1816.94 1540.2 0.0) (210 0.0 0.0 1.0))

(length en1) → 12

LISTP	指定された項目が、リストかどうか調べます

(listp '(100 100)) → T
(listp '100) → nil
(listp (+ 10 20)) → nil

TBLSEARCH	シンボル テーブル内のシンボル名を検索します

書式は：(tblsearch シンボルテーブル 検索するシンボル名)
①寸法スタイル [dimstyle] に <ISO-21> が有るかどうかを調べます。
(tblsearch "dimstyle" "ISO-21") → nil（無い場合）
②寸法スタイル [dimstyle] に <standard> が有るかどうかを調べます。
(tblsearch "dimstyle" "standard") → T の場合は、その情報を表示します。
（一例）
((0 . "DIMSTYLE") (2 . "Standard") (70 . 0) (3 . "") (4 . "") (5 . "") (6 . "") (7 . "") (40 . 1.0) (41 . 0.18) (42 . 0.0625) (43 . 0.38) (44 . 0.18) (45 . 0.0) (46 . 0.0) (47 . 0.0) (48 . 0.0) (140 . 0.18) (141 . 0.09) (142 . 0.0) (143 . 25.4) (144 . 1.0) (145 . 0.0) (146 . 1.0) (147 . 0.09) (71 . 0) (72 . 0)(73 . 1) (74 . 1) (75 . 0) (76 . 0) (77 . 0) (78 . 0) (170 . 0) (171 . 2) (172 . 0) (173 . 0) (174 . 0) (175 . 0) (176 . 0) (177 . 0) (178 . 0) (270 . 2) (271 . 4) (272 . 4) (273 . 2) (274 . 2) (340 . < 図形名 : 7ef01c88>) (275 . 0) (280 . 0) (281 . 0) (282 . 0) (283 . 1) (284 . 0) (285 . 0) (286 . 0) (287 . 3) (288 . 0))

AutoLISP の関数

TBLNEXT	シンボル テーブル内の次の項目を取得します

書式は：(tblnext　シンボルテーブル　[rewind])

①図面内の最初の画層を取得します。

(tblnext　"layer")　　→ AutoCAD は画層は必ず 1 つ持ちます。

((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))

戻り値の意味は、以下のとおりです。

(0 . "LAYER") シンボル タイプ

(2 . "0") シンボル名

(70 . 0) フラグ

(62 . 7) 色番号、非表示の場合は負数

(6 . "CONTINUOUS") 線種名

②次の画層を取得します。

(tblnext　"layer")　　→ nil（これ以上無い場合）

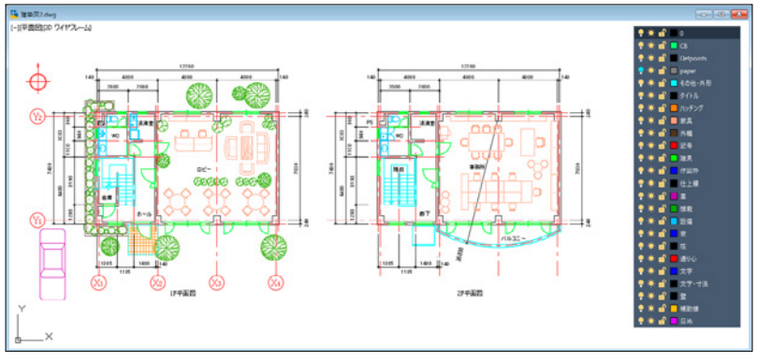
Point!

tblnext 関数は、繰り返し使用されるたびに、指定したテーブル内の次の項目を返します。rewind 引数が指定され、その値が nil 以外の場合は、シンボル テーブルの最初に戻り、シンボル テーブルの最初の項目が取得されます。

→②において、(tblnext　"layer"　T) とすると、返り値は <nil> ではなく、最初に戻り再度、<((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))> の値が返ります。

有効な シンボルテーブルには、"LAYER"、"LTYPE"、"VIEW"、"STYLE"、"BLOCK"、"UCS"、"APPID"、"DIMSTYLE"、"VPORT" があります。引数は、大文字と小文字は区別されません。

tbl_lay.lsp	図面の画層一覧を表示する
目的：	図面内の全画層名を取得し、テキスト スクリーンに表示します。
主要関数：	<tblnext><while><assoc><cdr><strcat><princ><textscr>



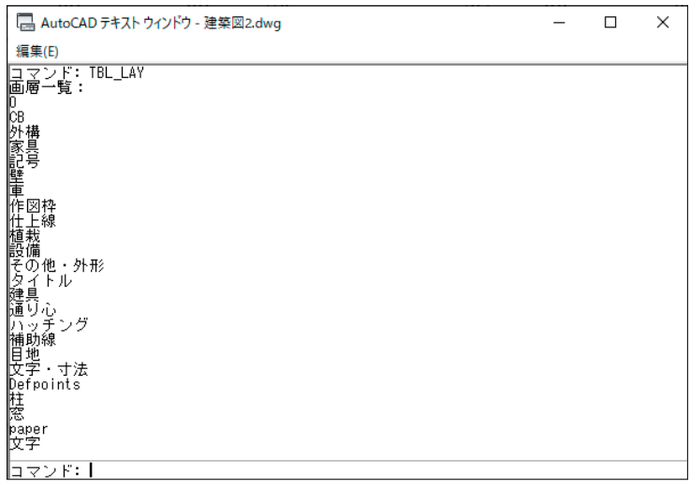

```
(defun C:tbl_lay( / lay_all layname laydata)
❶ (setq lay_all " 画層一覧：") ; テキスト 画面に表示する最初の表題を設定
❷ (setq layname (tblnext "LAYER" T)) ; 最初の画層の情報を取得する
❸ (while layname ; layname が nil を返すまで繰り返す
❹  (setq laydata (cdr (assoc 2 layname))) ;layname の画層名だけを取得する
❺  (setq lay_all (strcat lay_all "¥n" laydata)) ; 画層名を 1 つの文字列にする
❻  (setq layname (tblnext "LAYER")) ; 次の画層 ( 無ければ nil を返す )
 );while
❼ (princ lay_all) ; 画層名一覧の文字列を表示する
❽ (textscr) ; テキスト ウィンドウに切り替えて表示する
 (princ)
);end
```

while は、P1-129 を参考

strcat は、P1-67 を参考

textscr は、P1-140 を参考

テキスト ウィンドウに表示した画層一覧

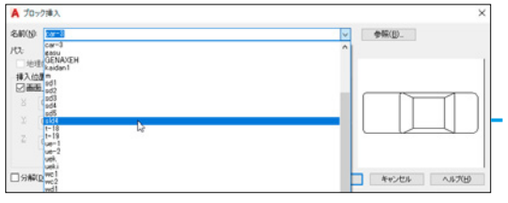


例題は、P2-149 を参考

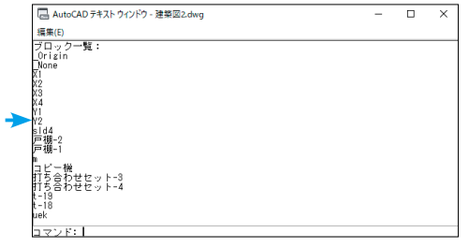
Point!

上記プログラムの②の (tblnext "LAYER" T) を (tblnext "BLOCK" T) に、⑥の "LAYER" を "BLOCK" に変更すると、下図のように、この図面にあるブロックの一覧を表示できます。

[ブロック挿入] ダイアログ



テキスト ウィンドウに表示したブロック一覧



第 2 節　算術演算関数

AutoLISP には、整数や実数を処理するための関数がありますが、その書式は通常の数式の記述方法とは異なります。
AutoLISP は CAD の業務に特化した LISP ですから、特に三角関数を初めとする角度の計算方法を理解することが重要です。
この節では、最も良く利用される基本的な算術演算関数を学びます。

使用頻度の高い AutoLISP の算術演算関数を、次の表に示します。（順不同）

①四則計算の算術演算関数	
関　数	説　明
(+ (add) [number number] ...)	すべての数値の合計を返します。
(- (subtract) [number number] ...)	1 番目の数値から 2 番目以降の数値を引いた差を返します。
(* (multiply) [number number] ...)	すべての数値の積を返します。
(/ (divide) [number number] ...)	1 番目の数値をそれ以外の数値の積で割った商を返します。
(1+ (increment) number)	数値を 1 増加します (インクリメントします)。
(1- (decrement) number)	数値を 1 減少します (デクリメントします)。
(exp number)	定数 e(実数) を指定された値で累乗した結果を返します (逆自然対数)。
(expt base power)	指定された値で数値をべき乗した結果を返します。
(sqrt number)	数値の平方根を実数で返します。
(log number)	数値の自然対数を実数で返します。

②角度の算術演算関数	
関　数	説　明
(atan num1 [num2])	数値のアークタンジェント (逆正接) をラジアン単位で返します。
(cos ang)	角度のコサイン (余弦) をラジアン単位で返します。
(sin ang)	ラジアン単位の実数で表した角度のサイン (正弦) の値を返します。

③最大・最小の算術演算関数	
関　数	説　明
(abs number)	引数の絶対値を返します。
(gcd int1 int2)	2 つの整数の最大公約数を返します。
(max [number number ...])	指定された数値の中の最大値を返します。
(min [number number ...])	指定された数値の中の最小値を返します。
(rem [num1 num2 ...])	1 番目の数値を 2 番目の数値で除算し、剰余を返します。

④論理の算術演算関数	
関　数	説　明
(~ (bitwise NOT) int)	引数のビット方式の NOT(1 の補数) を返します。
(logand [int int ...])	整数のリストのビット方式の AND(論理積) の結果を返します。
(logior [int int ...])	整数のリストのビット方式の OR(論理和) の結果を返します。
(lsh [int numbits])	指定されたビット数だけ整数を論理的にビット シフトした結果を返します。
(minusp number)	数値が負かどうかを調べます。
(zerop number)	数値がゼロに評価されるかどうかを調べます。

1　四則計算の演算関数

＋	引数の総和を返します

```
(setq a 10 b 20 c 30)
( + a b c)                    → 60
```

ー	2 番目以降の引数の総和を第 1 引数から引いた値を返します

```
(setq a 10 b 20 c 30)
( - a b c)                    → － 40 (10-20-30)
```

＊	全ての引数の積を返します

```
(setq a 10 b 20 c 30)
( * a b c)                    → 6000 (10 × 20 × 30)
```

/	1 番目の引数を 2 番目以降の引数の積で割った値を返します

```
(setq a 30.0 b 20.0 c 10.0)
( * a b c)                    → 0.15 (30 ÷ 20 ÷ 10)
```

1+	引数に 1 を加えた値を返します (1 と＋の間を空けてはいけません。)

```
(setq a 10 )
(setq a (1+ a )               → a = 11
```

1-	引数から 1 を引いた値を返します (1 とーの間を空けてはいけません。)

```
(setq a 10 )
(setq a (1- a )               → a = 9
```

EXPT	引数の乗を返します

```
(expt 2 3)                    → 8 (2 × 2 × 2)
(expt 2.0 4.0)                → 16.0 (2.0 × 2.0 × 2.0 × 2.0)
```

SQRT	実数の平方根を返します

```
(sqrt 9)                      → 3.0
(sqrt 3)                      → 1.73205
```

2 角度の演算関数

SIN	角度の正弦を返します（角度は、ラジアンで指示します。）

(sin 0.0) → 0.0
(sin (/ pi 2)) → 1.0
(sin pi) → 1.22465e-016

COS	角度の余弦を返します（角度は、ラジアンで指示します。）

(cos 0.0) → 1.0
(cos (/ pi 2)) → 6.12323e-017
(cos pi) → -1.0

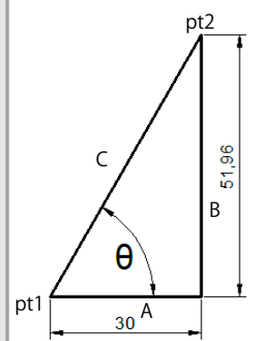
ATAN	数値のアークタンジェント (逆正接) をラジアン単位で返します

書式：(atan num1 num2)
num1 引数のみを指定した場合は、num1 のアークタンジェント (ラジアン単位)。
num1 と num2 引数の両方を与えた場合、atan 関数は、num1/num2 のアークタンジェントをラジアン単位で返します。
(atan 1.0) → 0.785398
(atan -1.0) → -0.785398
(atan 2.0 3.0) → 0.588003
(atan 2.0 -3.0) → 2.55359

Point!

線分の極座標入力は、直線距離と角度を入力します。
右図のように、点 pt1 から点 pt2 へ線分を作成するために必要な情報は、直線距離（C）と角度 (θ) です。
今、X 軸の距離 (A) と Y 軸の距離 (B) が与えられた時（直交座標）、これから、直線距離（C）と角度 (θ) の値を求めることになります。

①直線距離は、$C^2 = A^2 + B^2$ より、C (pt1-pt2) を求めます。
②角度 (θ) を求める関数は、ATAN 関数になります。
(atan 51.96 30.0) → 1.04718（ラジアン）が返ります。
180° ： 3.14（ラジアン）＝θ：1.04718（ラジアン）から、
θ ＝60° になります。



ラジアンを角度の文字列に変換する関数 (angtos) を使うと以下のようになります。
(angtos (atan 51.96 30.0)) → "60"

3 最大・最小の演算関数

MAX	最大の値を返します

(setq a 10 b 20 c 30)
(max a b c) → 30

MIN	最少の値を返します

(setq a 10 b 20 c 30)
(min a b c) → 10

ABS	引数の絶対値を返します

(abs -65.4) → 65.4
(abs 3.14) → 3.14

REM	割り算の余りを返します（第１引数を第２引数で割った余り。）

(rem 30 4) → 2
(rem 3 4.0) → 3.0

4 論理の演算関数

minusp	数値が負かどうかを調べます

数値が負の時はＴ、それ以外は nil を返します。
(minusp -10.0) → T
(minusp 10.0) → nil

zerop	数値がゼロに評価されるかどうかを調べます

評価結果が 0(ゼロ）の時はＴ、それ以外は nil を返します。
(zerop 0.0) → T
(zerop 0.001) → nil

第 3 節　文字処理関数

① AutoCAD が提供する文字処理関係のコマンド

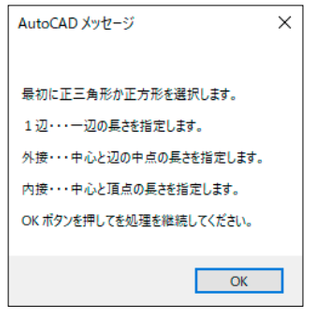
AutoCAD には、文字列を処理するための関数があります。下図は、[文字検索] コマンドを実行したときに表示される [検索と置換] のダイアログ ボックスです。

このダイアログには、文字処理をコントロールする多くの機能が提供されています。

②ユーザーが作成した文字処理関係の AutoLISP（第 2 部 3 章 8 節より）

下図はユーザーが作成したヘルプのコメントです。ダイアログの [Help] ボタンを押すと解説文を表示します。左側のように記述すれば、右のダイアログが表示されます。

```
(strcat
   "¥n 最初に正三角形か正方形を選択します。"
   "¥n  "
   "¥n 1 辺・・・一辺の長さを指定します。"
   "¥n  "
   "¥n 外接・・・中心と辺の中点の長さを指定します。"
   "¥n  "
   "¥n 内接・・・中心と頂点の長さを指定します。"
   "¥n  "
   "¥nOK ボタンを押してを処理を継続してください。"
)
```



この節では、AutoCAD の組み込みコマンドと同様の文字列の処理の仕方を学びます。

使用頻度の高い AutoLISP の 文字列処理関数を、次の表に示します。（順不同）

文字列処理関数	
関　数	説　明
(strcase string [which])	すべてのアルファベットを大文字または小文字に変換した文字列を返します。
(strcat [string1 [string2] ...)	複数の文字列を連結した文字列を返します。
(strlen [string] ...)	文字列の文字数を示す整数を返します。
(substr string start [length])	文字列の部分文字列を返します。
(read [string])	文字列から取得した最初のリストまたはアトムを返します。
(wcmatch string pattern)	ワイルドカードを使用して、文字列のパターン マッチングを行います。

1 主な文字列処理関数

STRCASE　英字の大文字と小文字を変換します

(strcase "AutoLISP")　→ AUTOLISP （全部大文字になります。）
(strcase "AutoLISP" T)　→ autolisp （全部小文字になります。）

文字列内のすべてのアルファベット文字 (全角文字を含む) を大文字または小文字に変換します。この関数には 2 つの引数を指定します。文字列を大文字で返すか小文字で返すかを指定するオプションの引数です。2 番目の引数を指定しないと、その引数は nil に評価され、strcase は大文字に変換された文字列を返します。

STRCAT　文字列を結合します

(strcat "Auto" "LISP")　→ "AutoLISP"
(strcat "@" "100" "<" "45")　→ "@100<45"（極座標入力で使用します。）

複数の文字列を 1 つの文字列に結合します。これは、通常の文字列の中に変数文字列を挿入するときに便利です。

STRLEN　文字列の長さを返します

(strlen "AutoLISP")　→ 8
(strlen "123 456")　→ 7（空白も 1 文字と数えます。）
(strlen " 文字列 ")　→ 6（日本語は 2 バイトです。）
(strlen)　→ 0
(strlen "")　→ 0
(strlen "ABC" "123")　→ 6（引数の文字列が複数あると、その合計を返します。）

SUBSTR　文字列の指定した番目から指定した数の文字を返します

(substr "AutoLISP" 5 2)　→ "LI" （1 から数えます。）
(substr "AutoLISP" 5)　→ "LISP"（個数を省略すると、最後の文字まで返します。）

substr 関数は、文字列の部分文字列 (サブストリング) を返します。この関数には、2 つの必須引数と 1 つの省略可能な引数を指定します。最初の引数には、文字列を指定します。
2 番目の引数には、部分文字列に含める文字列の先頭の文字の位置を指定する正の整数です。
3 番目の引数を指定する場合は、部分文字列の文字数を指定します。
3 番目の引数を指定しないと、substr は指定した開始文字以降のすべての文字を返します。

moji_len.lsp	文字列の中にある空白文字を除外する
目的：	このプログラムは、コマンドラインに貼り付けられた文字列から、文字列に含まれる空白文字を除外します。
主要関数：	<getstring><strlen><repeat><substr><strcat><alert>

Step1 ー (getstring T "¥n コマンドラインに文字列を貼り付けてください：")) ❶

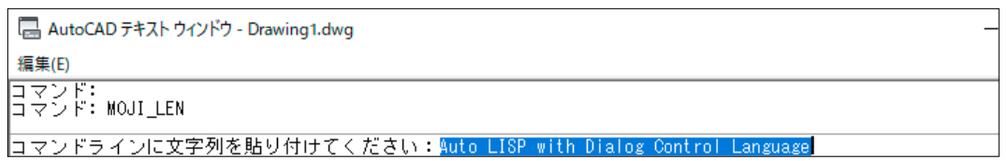
" コマンドラインに文字列を貼り付けてください：" のメッセージを表示して、コマンドラインへの英数字の貼り付けを受け入れます。（空白のある文字列）

(getstring T　"¥n メッセージ ")

→ getstring の後に <T> を入れると、空白 < スペース > を受け入れます。もし <T> を付けないと、スペースキーを押した時点で終了してしまいます。

（<T> 以外の英数字でも構いません。）

（コマンドラインに "Auto LISP with Dialog Control Language" の文字を貼り付けた例。）



Step2 ー ① (strlen moji_old) ❷

strlen 関数で文字の長さを取得します。英数字は <1> ですが、日本語は <2> を返します。

② (moji_new "") ❸

新しい文字列の初期値を "" (空) にセットします。repeat 関数で順番に文字を追加します。

Step3 ー ① (setq ｉ 1) ❹

substr 関数で文字を順番に取り出していくカウントを <1> にセットします。

② (repeat moji_len　・・・) ❺

文字の個数分だけ以下のプログラムを繰り返します。

③ (substr moji_old i 1) ❻

入力した文字列から 1 つ読み込んで変数 <moji> に代入します。

substr 関数の書式は、(substr 文字列 何番目 文字数) です。

④ (if (/= moji " ") ❼

読み込んだ文字が、<" "> でなければ、次の 1 行目の作業を行います。

⑤ (strcat moji_new moji) ❽

取得した文字を最初に用意した文字列 <moji_new> と結合します。

strcat 関数の書式は、(strcat 元の文字列 追加する文字) です。

⑥次の文字を取り出すため、カウント <i> を 1 つ増やして、⑤に戻ります。❾

Step4 ー (setq ans (strcat ・・) ❿

コマンドラインに貼り付けた元の文字列と空白を取り除いた新しい文字列を、ダイアログとコマンドラインに表示する文字を連結します。

<¥n> は改行コードです。

Step5 ー① (alert ans_moji) ⓫

元の文字列と空白を取り除いた文字列をダイアログに表示します。

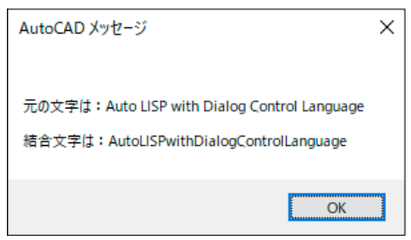
② (princ ans_moji) ⓬

元の文字列と空白を取り除いた文字列をコマンドラインに表示します。

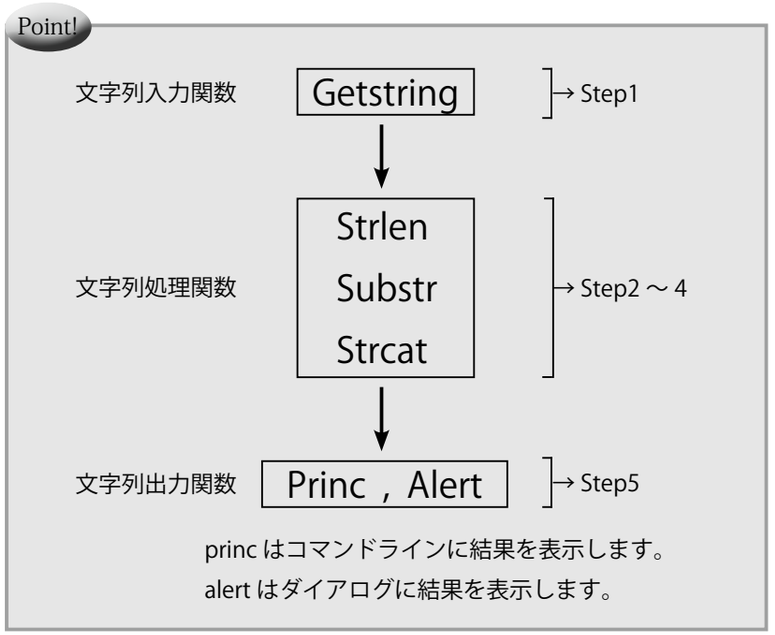
③ (princ) ⓭

コマンドラインに結果を表示させません。

Step5 ー①の結果



Step5 ー②の結果



princ は、P1-141 を参考
alert は、P1-198 を参考

```
(defun C:moji_len( / moji_old moji_len moji_new i moji ans_moji)
 ; クリップボードからコマンドラインに空白のある英数字の文字列を貼り付ける
❶ (setq moji_old(getstring T "¥n コマンドラインに文字列を貼り付けてください : "))   → Step1
❷ (setq moji_len (strlen moji_old))  ; 入力された文字数を取得                          → Step2
❸ (setq moji_new "")  ; 空白を取り除いた文字列の初期値を <""> にセット
 ;repeat 関数で文字列から 1 つずつ文字を取り出し、空白を取り除いて文字を再度連結する
❹ (setq i 1)  ;substr 関数で文字を順番に取り出していくカウントを <1> にセット
❺ (repeat moji_len  ; 文字の数だけ繰り返す
❻   (setq moji (substr moji_old i 1))  ; 文字を 1 つずつ取り出す
❼   (if (/= moji " ")  ; 文字が <" ">（1 文字空白）でなければ、以下の処理を行う
❽     (setq moji_new (strcat moji_new moji))  ; 新しい文字列に追加
     ) ;if
❾   (setq i (1+ i))  ; 次の文字を取り出すため、カウント <i> を 1 つプラスして⑤に戻る   → Step3
   ) ;repeat
 ; コマンドラインとダイアログに表示するために文字を 1 つに連結する
❿ (setq ans_moji (strcat                                                              → Step4
                   "¥n 元の文字は：" moji_old
                   "¥n "
                   "¥n 結合文字は：" moji_new
                  );strcat
 );setq
⓫ (alert ans_moji)  ; ダイアログに元の文字列と新しい文字列を表示する                   → Step5
⓬ (princ ans_moji)  ; コマンドラインに元の文字列と新しい文字列を表示する
⓭ (princ)  ; コマンドラインに結果を表示させない
 ) ;end
```

番号	関数名	説　明
①	getstring T	getstring の後に <T> を入れると、空白 < スペース > も受け入れます。
②	strlen	文字の長さを取得する関数です。
③	(setq moji_new "")	新しい文字の初期値を ""（空）にセットして、スタートします。
④	(setq i 1)	文字を順番に取り出していくカウントを <1> にセットします。
⑤	repeat	文字の個数分だけ次のプログラムを繰り返します。
⑥	substr	取得した文字列から 1 つずつ読み込みます。
⑦	(if (/= moji " ")	読み込んだ 1 文字が、<" "> かどうかをチェックします。
⑧	strcat	取得した文字列を順番に連結していきます。
⑨	(setq i (1+ i))	次の文字を取り出すため、カウント <i> を 1 つ増やして⑤に戻ります。
⑩	strcat	結果を表示するために、元の文字列と新しい文字列を連結します。
⑪	alert	連結した文字列をダイアログに表示します。
⑫	princ ans_moji	連結した文字列をコマンドラインに表示します。
⑬	princ	⑫の結果を再表示しません。

☞getstring は、P1-153 を参考
repeat は、P1-124 を参考

READ	文字列から取得した最初のリストまたはアトムを返します

(read "100 200 300") → 100 [整数値]
(read " 建具 外構 寸法線 ") → 建具 [シンボル]（注：文字列では無い）
(read "(建具 外構 寸法線)") → (建具 外構 寸法線) [最初のリスト]
(read "(a b c)(c d)") → (a b c) [最初のリスト]

read 関数は任意の LISP データの文字列を分析して、文字列内の最初の式を適切なデータタイプに変換して返します。

WCMATCH	ワイルドカードを使用して、文字列のパターン マッチングを行います

書式　(wcmatch　比較する文字列　マッチングするパターンを含んだ文字列)
2 つの引数がマッチしたときは、<T> を返します。しなかったときは、<nil> を返します。
比較するときは、大文字と小文字を区別します。

(wcmatch "LISP" "Lisp") → nil
(wcmatch "LISP" "LISP") → T
(wcmatch "LISP" "L*") → T

比較する文字列が複数あるときは、その中のいずれかを満足していれば <T> を返します。
(wcmatch "LISP" "Lisp,L???") → T（2 番目にマッチしている）

円記号 <¥> の検索（<¥> 記号はエスケープ文字として使用されるので、文字列内で <¥> 記号を作成するには、2 つの円記号 <¥¥> が必要です。）
(wcmatch "LISP" "?¥¥?) → nil

ワイルド カード文字	
文字	定　義
# (シャープ記号)	1 つの数字にマッチします
@ (アット記号)	1 つのアルファベット文字にマッチします
. (ピリオド)	1 つの非英数字にマッチします
* (アスタリスク)	空文字列を含む任意の文字シーケンスにマッチし、検索パターンの先頭、中間、末尾のどこにでも使用できます。
? (クエスチョン マーク)	任意の 1 文字にマッチします
~ (ティルダ)	パターンの先頭の文字の場合、パターン以外の任意の文字にマッチします
[...]	囲まれた文字のいずれか 1 文字にマッチします
[~...]	囲まれた文字以外の 1 文字にマッチします
- (ハイフン)	角括弧内で使用され、文字の範囲を指定します
, (カンマ)	2 つのパターンを区切ります
` (逆クォーテション)	特殊文字をエスケープします (次の文字を文字どおりに読みます)

AutoLISP の関数

第 4 節　ジオメトリック関数

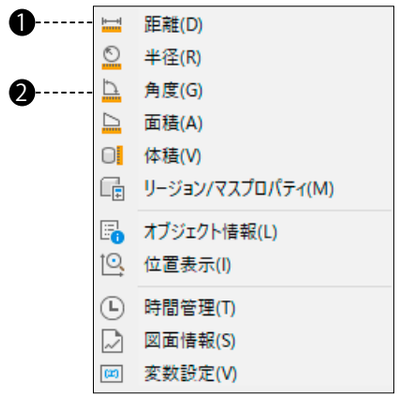
下図は [情報] の中にある [ジオメトリック計算] を行うメニューです。

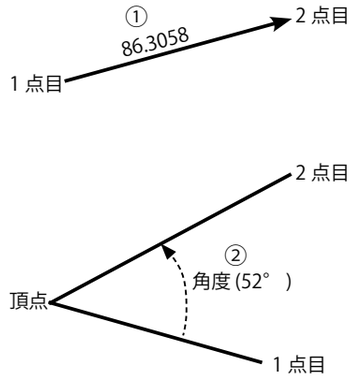
①の [距離] を選択すると、次のようなメッセージが表示されます。

1 点目を指定 :
2 点目を指定 または [複数点 (M)]: 長さ = 86.3058、　XY 平面の角度 = 2、XY 平面からの角度 = 0
デルタ X = 86.2691、デルタ Y = 2.5166、デルタ Z =　0.0000

②の [角度] を選択すると、次のようなメッセージが表示されます。

角度の頂点を指定 :
頂点からの角度の 1 点目 :
頂点からの角度の 2 点目 :
角度 = 52°





上記の①を AutoLISP では以下のように記述します。
　(distance　1 点目　2 点目)
同様に②を AutoLISP では以下のように記述します。
　(angle　1 点目　2 点目)

この節では、このようにオブジェクトの長さや角度などの計算方法を学びます。

使用頻度の高い AutoLISP のジオメトリック関数を、次の表に示します。

①距離のジオメトリック関数	
関　数	説　明
(distance pt1 pt2)	2 点間の 3D 距離を返します。

②角度のジオメトリック関数	
関　数	説　明
(angle pt1 pt2)	2 つの端点で定義される線分の角度をラジアン単位で返します。
(polar pt ang dist)	指定された点から指定された角度と距離だけ離れた UCS 3D 点を返します。

③座標計算のジオメトリック関数	
関　数	説　明
(inters pt1 pt2 pt3 pt4 [onseg])	2 本の線分の交点を検出します。
(osnap pt mode)	指定された点にオブジェクト スナップ モードを適用して取得した 3D 点を返します。
(textbox elist)	指定された文字オブジェクトを計測し、その文字を囲むボックスの対角頂点の座標を返します。

1　距離のジオメトリック関数

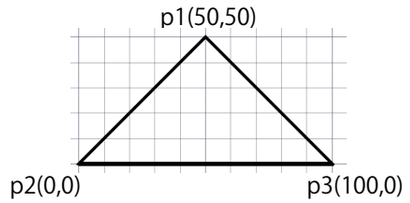
DISTANCE	既知の 2 点間の距離を返します。

```
(setq  p1  (getpoint  "¥n1 点目を指示 : "))
(setq  p2  (getpoint  p1 " ¥n2 点目を指示 : "))
(setq  dis1 (distance  p1  p2))
```

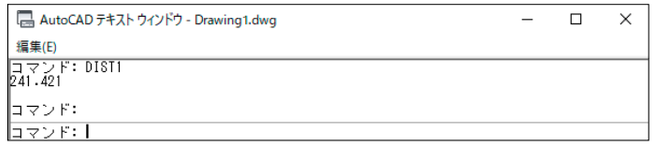
コマンド : !dis1　　→ コマンドラインに !dis1 と入力します。
694.475　　→ コマンドラインに距離が表示されます。

distance 関数	三角形の三辺の長さを合計する

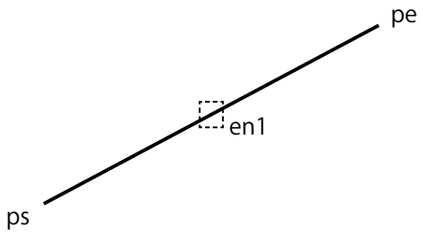
三角形の 3 点を選択して、三辺の長さの合計を計算します。




```
(defun  C:dist1( / p1 p2 p3 dis1 dis2 dis3 dis_sum)
  (setq  p1  (list  50  50))          → p1 の座標を (50,50) にセットします。
  (setq  p2  (list  0  0))            → p2 の座標を (0,0) にセットします。
  (setq  p3  (list  100  0))          → p3 の座標を (100,0) にセットします。

  (setq  dis1 (distance  p1 p2))      → p1 から p2 の長さを計算します。
  (setq  dis2 (distance  p2 p3))      → p2 から p3 の長さを計算します。
  (setq  dis3 (distance  p3 p1))      → p3 から p1 の長さを計算します。

  (setq  dis_sum (+ dis1  dis2  dis3))  → 三辺の長さの合計を計算します。
  (princ  dis_sum)(princ)             → コマンドラインに表示します。
) ; end                                 <241.421> が表示されます。
```



dist2.lsp	選択した線分の長さを計算する
目的：	線分を選択し、その図形情報から始点と終点の座標を取得して、線分の長さを計算します。
主要関数：	<entsel><car><entget><assoc><cdr><distance>



Step1 ー マウスで既存の線分を選択します。(en1) ❶

コマンド : (setq en1 (entsel "¥n 線分を指示 : "))
線分を指示 : (< 図形名 : 7ef03998> (314.077 600.574 0.0))

→ entsel 関数は、図形名と選択した位置の座標値を取得します。

Step2 ー 図形情報から、図形名だけを取得します。(en2) ❷

コマンド : (setq en2 (entget (car en1)))
((-1 . < 図形名 : 7ef03998>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A3") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 233.691 566.153 0.0) (11 539.923 711.483 0.0) (210 0.0 0.0 1.0))

→ (car en1) から図形名 <7ef03998> が取得できます。
（en1 は図形名と座標値の組み合わせ < ドット・ペア > なので、car 関数でリストの先頭を取得します。）
→ entget 関数は、図形の情報を開示します。（これを変数 <en2> にセットします。）

Step3 ー選択した線分の始点 (ps) の座標を取得します。❸

コマンド : (setq ps (cdr (assoc 10 en2)))
(233.691 566.153 0.0)

→ (assoc 10 en2) から始点 (10 233.691 566.153 0.0)<10 と座標のセット > を取り出します。
→ cdr 関数で 2 番目以降（X、Y、Z 座標）を取り出します。

Step4 ー 選択した線分の終点 (pe) の座標を取得します。❹

コマンド : (setq pe (cdr (assoc 11 en2)))
(539.923 711.483 0.0)

→ (assoc 11 en2) から終点 (11 539.923 711.483 0.0)<11 と座標のセット > を取り出します。
→ cdr 関数で 2 番目以降（X、Y、Z 座標）を取り出します。

Step5 ー distance 関数で、線分の長さ (dis1) を計算します。❺

コマンド : (setq dis1 (distance ps pe))
→ 338.968

Step6 ー princ 関数で、結果を表示します。❻

Step7 ー princ 関数だけの場合は、コマンドラインには、何も表示しません。❼

```
(defun C:dist2( / en1 en2 ps pe dis1)
❶    (setq en1 (entsel "¥n 線分を指示 : "))  ; 線分の指示          → Step1
❷    (setq en2 (entget (car en1)))           ; 図形情報を en2 に代入  → Step2
❸    (setq ps (cdr (assoc 10 en2)))          ; 始点の座標を取得       → Step3
❹    (setq pe (cdr (assoc 11 en2)))          ; 終点の座標を取得       → Step4
❺    (setq dis1 (distance ps pe))            ; 始点と終点の距離を計算  → Step5
❻    (princ dis1)                            ; 結果を表示             → Step6
❼    (princ)                                                          → Step7
) ;end
```

番号	関数名	説　明
①	entsel	ユーザーにオブジェクトを選択させます。図形名と座標値が取得できます。
②	entget	図形名から図形情報を取得します。
③	cdr(assoc 10)	(assoc 10) の 10 は始点のグループ コードです。その 2 番目以降が始点の座標です。
④	cdr(assoc 11)	(assoc 11) の 11 は終点のグループ コードです。その 2 番目以降が終点の座標です。
⑤	distance	2 点間の距離を計測する関数です。
⑥	princ dis1	princ の後に距離の数値を表示します。
⑦	princ	最後の行に、<nil> を表示させません。

entsel は、P1-210 を参考
entget は、P1-218 を参考

AutoLISP の関数

2 角度のジオメトリック関数

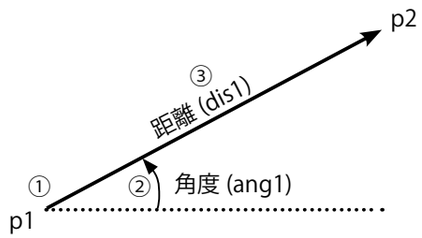
ANGLE	既知の 2 点間の角度を返します。（返り値はラジアン）

(setq p1 (getpoint "¥n1 点目を指示："))
(setq p2 (getpoint p1 " ￥n2 点目を指示："))
(setq ang1 (angle p1 p2))

コマンド : !ang1　　→ コマンドラインに ang1 と入力します。
0.646632　　→ コマンドラインに角度が表示されます。

POLAR	基点からの角度（ラジアン）と距離で次点の座標を返します。

基点 <p1> からの角度 <ang1> と距離 <dis1> を入力して、次点 <p2> を求めます。
(polar 基点 角度 距離)



(setq p1 (getpoint "¥n 基点を指示："))　　; ① p1 の座標を取得する
(setq ang1 (getangle "¥n 角度を指示："))　　; ②の角度を取得する (ラジアン)
(setq dis1 (getdist "¥n 距離を指示："))　　; ③の距離を取得する
(setq p2 (polar p1 ang1 dis1))　　; ①②③を引数として、polar 関数で p2 を計算する
(command "LINE" p1 p2 "")　　; 線分コマンドで p1 から p2 の線分を作図する

Point!

polar 関数で使用する角度はラジアンです。
②の角度を取得するときに [getangle] 関数ではなく [getreal] 関数で数値を取得した場合は、度数を角度のラジアンに変換する処理が必要になります。
十進数→ラジアンへ変換するコード (* pi (/ 度数 180.0)) を追加します。

(setq ang1 (getreal "¥n 角度を入力：")) ; 角度を度数で取得
(setq ang1 (* pi (/ ang1 180.0)))　　; 度数をラジアンに変換する

度数からラジアンへの変換は、P1-78、1-79 を参考

①角度を使った関数　・・・　angle

angle 関数は、すでに分かっている 2 点間の角度を計算します。返される値はラジアンで表示されます。

0 度	0 ラジアン
45 度	0.25 π ラジアン (約 0.785 ラジアン)
90 度	0.5 π ラジアン (約 1.57 ラジアン)
180 度	π ラジアン (約 3.14 ラジアン)
360 度	2 π ラジアン (約 6.28 ラジアン)

（例）　点 A(0,0) と点 B(100,100) の角度 (Ang1) を求めてみましょう。

(setq A (list 0.0 0.0))
(setq B (list 100.0 100.0))
(setq Ang1 (angle A B))

上記の 3 行をコマンドラインに入力すると、コマンドラインに 0.785398 が表示されます。

②角度を使った関数　・・・　polar

線分コマンドで、始点を指示した後、終点をキーボードから入力するときに、始点からの長さと角度で終点の座標を指示することもあります。（極座標入力）
このときに、使用される関数が Polar 関数です。

書式は次のようになります。　(polar 基点 角度 距離)
ただし、この場合の角度はラジアンです。
線分コマンドで終点の位置が、始点からみて、長さが 100、角度が 30 度の場合、次のようにキーボード入力します。　@100<30
しかし、これをプログラムで記述するときは、30 度をラジアンに換えなければいけません。
180 度が pi(πの記号はキーボードには無いので、pi と表します。) ですから、30 度は (/ pi 6) になります。
すなわち、プログラムでは、(polar 基点 (/ pi 6) 100) になります。

（例）　最初の点をユーザーにマウスで指示 (P1) させて、二点目をその点から、長さ 100、角度 30 度の位置の座標 (P2) を求めるには次のように記述します。

(setq P1 (getpoint “¥n 始点を指示：”))
(setq P2 (polar P1 (/ pi 6) 100.0))
(command “LINE” P1 P2 “”)

度数 (degree) からラジアン (radian) への変換

前ページの polar 関数ではラジアンが使われます。しかし、ユーザーは度数で入力する場合もありますから、プログラムを作成する場合は、そのプログラムの中でこの変換作業をしなければいけません。

次のケース①とケース②のコードを比べて見ましょう。
ケース①では、2 行目の角度の取得に [getangle] を使っています。これはラジアンを取得しますからこのコードは正しく動きます。

ケース①

```
(setq  dist1 (getdist "¥n 距離を入力：" ))
(setq  ang1 (getangle  "¥n 角度を入力：" ))
(setq  P1    (getpoint  "¥n 始点を指示：" ))
(setq  P2    (polar  P1  ang1  dist1))
(command "LINE"  P1  P2  "")
```

ケース②

```
(setq  dist1 (getdist "¥n 距離を入力：" ))
(setq  ang1 (getreal  "¥n 角度を入力：" ))
(setq  P1    (getpoint  "¥n 始点を指示：" ))
(setq  P2    (polar  P1  ang1  dist1))
(command "LINE"  P1  P2  "")
```

しかし、ケース②ではエラーになります。2 行目の [getreal] は実数を取得する関数ですが、4 行目の ang1 に入るのはラジアンでなければいけません。
つまり、[getreal] を使うときは、ang1 をラジアンに変換した数値を入れなければいけないのです。

①角度を度数 (ang1) からラジアン (rad1) への変換
(* pi (/ ang1 180.0))

②角度をラジアン (rad1) から度数 (ang1) への変換
(* (/ rad1 pi) 180.0)

Point!

従って、ケース②は以下のように 3 行目を追加します。

```
(setq  dist1 (getdist "¥n 距離を入力：" ))
(setq  ang1 (getreal  "¥n 角度を入力：" ))
(setq  ang1 (* pi (/ ang1 180.0)))
(setq  P1    (getpoint  "¥n 始点を指示：" ))
(setq  P2    (polar  P1  ang1  dist1))
(command "LINE"  P1  P2  "")
```

Point!

①度数 (ang1) からラジアン (rad1) への変換式	②ラジアン (rad1) から度数 (ang1) への変換式
180°：pi(3.14) ＝ ang1：rad1 より	180°：pi(3.14) ＝ ang1：rad1 より
rad1 = (pi × ang1) / 180 が成立	ang1 = (180 × rad1) / pi が成立
これを LISP 式で表すと、	これを LISP 式で表すと、
解 1 rad1 = (* pi (/ ang1 180.0))	解 1 ang1 = (* (/ rad1 pi) 180.0)
解 2 rad1 = (/ (* pi ang1) 180.0)	解 2 ang1 = (/ (* 180.0 rad1) pi)

先ほどのプログラムに、この関数を混在させる場合、以下のようにします。

```
(defun dtr(a)
      (* pi (/ a 180.0))
)
```

ang.lsp の前に記述します。

```
(defun C:ang( / dist1 ang1 P1 P2)
(setq  dist1 (getdist "¥n 距離を入力：" ))
(setq  ang1 (getreal  "¥n 角度を入力：" ))
(setq  ang1 (dtr  ang1))
(setq  P1    (getpoint  "¥n 始点を指示：" ))
(setq  P2    (polar  P1  ang1  dist1))
(command "LINE"  P1  P2  "")
)
```

上記プログラムの dtr(a) 関数の ( ) には引数が 1 つ必要です。
(例) dtr(90.0)　　→ 1.5708 が返ります。

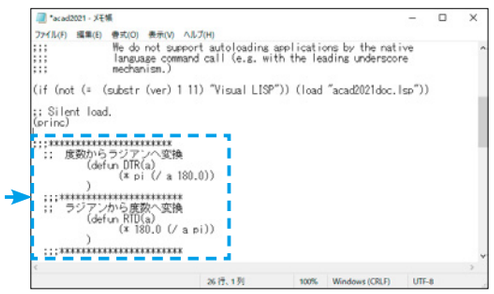
しかし、この場合のようにすべてのプログラムに同じ関数を記述するのは、効率がいいとは言えません。そこで、よく使用する関数を 1 つにまとめておいて、そこから参照するようにしたほうが効率よくなります。
AutoCAD はそのために acad.lsp と acaddoc.lsp を用意してあります。
acda.lsp に記述されている関数は、新規ファイルがロードされるごとに、メモリに読み込まれます。
しかし、acaddoc.lsp に記述されている関数は、一度読み込まれると他の図面にも有効です。

P1-8,9 を参考

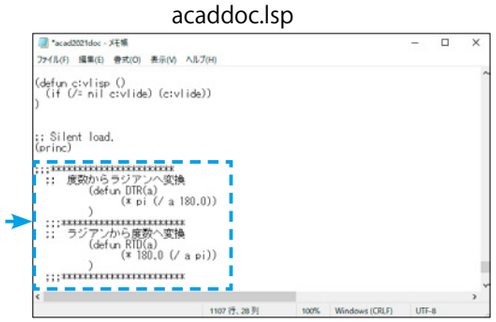
acad.lsp 又は acaddoc.lsp に書き入れた例

```
;;;*************************
;;   度数からラジアンへ変換
          (defun DTR(a)
                    (* pi (/ a 180.0))
          )
;;;*************************
;;   ラジアンから度数へ変換
          (defun RTD(a)
                    (* 180.0 (/ a pi))
          )
;;;*************************
```

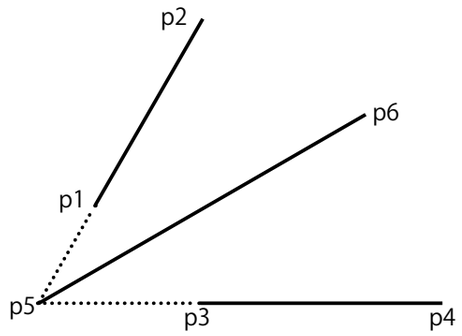
acad.lsp



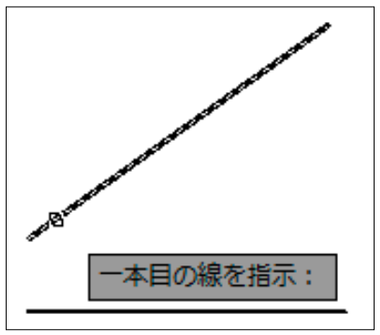
acaddoc.lsp



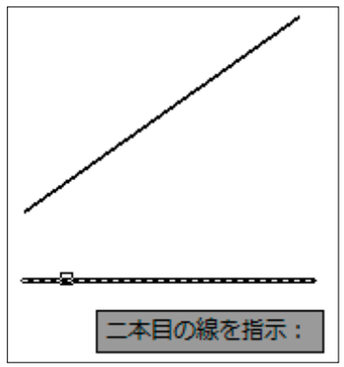
nitoubun.lsp	二等分線を作図する
目的：	線分 (p1-p2) と線分 (p3-p4) の交点 (p5) から、2 本の線分の二等分線 (p5-p6) を作図します。
主要関数：	<entsel><assoc><trans><inters><distance><angle>



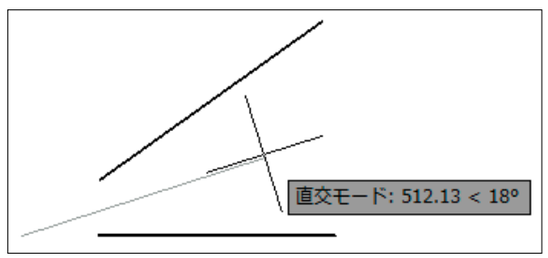
Step1 ー 1 本目の線分を選択します。



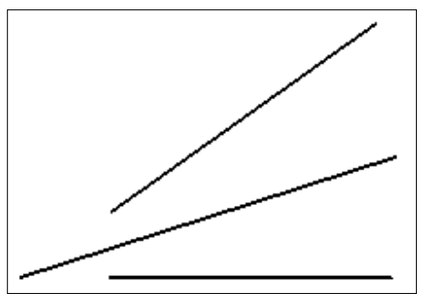
Step2 ー 2 本目の線分を選択します。



Step3 ～ 8 ー 2 等分角の線分を作成します。



Step9 ー最後に座標系等を元に戻します。



線分のグループ コード	
グループコード	意　味
100	サブクラス マーカー (AcDbLine)
39	厚さ (省略可能、既定 = 0)
10	始点 (WCS)　DXF: X 値、APP: 3D 点
20,30	DXF: 始点の Y および Z の値 (WCS)
11	終点 (WCS)　DXF: X 値、APP: 3D 点
21,31	DXF: 端点の Y および Z の値 (WCS)
210	押し出し方向 (省略可能、既定 = 0, 0, 1)　DXF: X 値、APP: 3D ベクトル
220,230	DXF: 押し出し方向の Y および Z の値 (省略可能)

Step1 ー 線分 (p1-p2) の始点と終点の座標を求めます。

① (setq ob1 (entsel " 一本目の線を指示：")) ❶

(< 図形名 : 7ef03990> (1387.56 1667.17 0.0))　→ 図形名と指示した位置の座標を取得

② ob1 から図形名だけを取り出します。

ob1 は図形名と座標値のリストですから、1 番目の図形名を取り出す関数は、car です。

(car ob1)　→ < 図形名 : 7ef03990> を取得

③ entget 関数で、< 図形名 : 7ef03990> の中身を開示します。

((-1 . < 図形名 : 7ef03990>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A2") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 1301.34 1642.56 0.0) (11 1784.79 1802.54 0.0) (210 0.0 0.0 1.0))

④始点の座標は (10 1301.34 1642.56 0.0) の中に格納されていますので、

assoc 関数で、このリストを取り出します。　→ (10 1301.34 1642.56 0.0)

⑤ 10 の次からのリストが X、Y、Z 座標です。❷

cdr 関数で、リストの 2 番目以降を取り出します。→ (301.34 1642.56 0.0) を取得

⑥終点の座標は、 (11 1784.79 1802.54 0.0) ですから同様にして取り出します。❹

⑦始点と終点の座標をワールド座標からユーザー座標に変更します。❸❺

Step2 ー同様にして、線分 (p3-p4) の始点と終点の座標を求めます。❻❼❽❾❿

ワールド座標系とユーザー座標系が同じ場合は、変更する必要はありません。しかし、事前に変更されていることもありますから、そのような時でもエラーにならないような処置をしておくことが大切です。

ユーザー座標系は、P1-110 を参考

Step3 ー線分 (p1-p2) と線分 (p3-p4) の交点を求めます。⓫

最後の引数に <nil> を加えると、2 本の線分が実際に交差していなくても仮想交点の座標を計算します。

(setq p5 (inters p1 p2 p3 p4 nil))

Step4 ー交点 p5 から遠い方の端点を p2 と p4 にします。⓬⓭⓮⓯

線分 p1-p2 と交点 p5 の距離を比較して、p1 より p2 が p5 より遠ければそのまま、p2 の方が p1 より p5 に近ければ、p1 と p2 の座標を入れ替えます。
同様に線分 p3-p4 で、交点 p5 から遠い方を p4 にします。

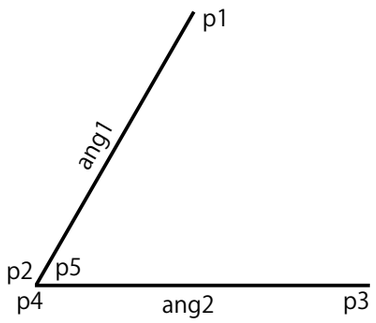
次の Step5 で交点 p5 から点 p2 への角度を求めるのですが、もし p5 と p2 が同じ座標（同様に p5 と p4 が同じ座標）であれば、角度が計算できません。（下図①のケース）
このエラーを避けるために、角度を計算する各点は必ず離れている点を指定する必要があります。（下図②のケース）

交点 p5 と線分 (p1-p2) の各点の座標の距離を比較して、p5 から遠くにある方の点を p2 にしています。線分 (p3-p4) も同様の処理をします。

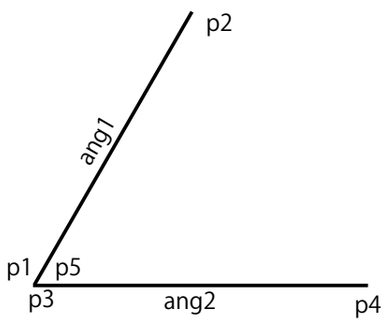
```
(if (> (distance p5 p1) (distance p5 p2))
   (setq s p1 p1 p2 p2 s))
```

もし、p1 が p2 より p5 から遠くにあれば、p1 と p2 を入れ替えます。
まず、p1 の座標を仮の変数 <s> にセットして、p2 の座標を p1 に移し、最後に仮の変数 <s> の座標を p1 に移します。
線分 (p3-p4) も同じ処理を行います。

① p5 と p2、p4 が同じ座標の場合（エラーになる）



② p5 より遠い方を p2 と p4 にする



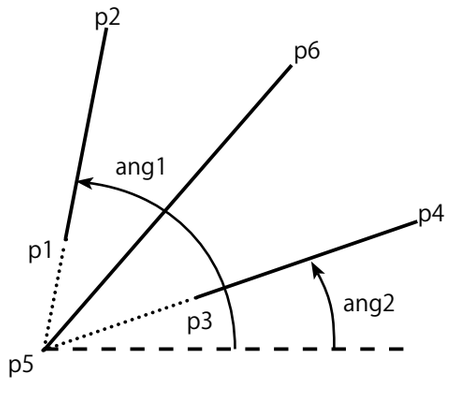
Step5 ー交点 (p5) から端点 p2 への角度を求めます。⓰⓱

(setq ang1 (angle p5 p2))
同様に、交点 (p5) から端点 p4 への角度を求めます。
(setq ang2 (angle p5 p4))

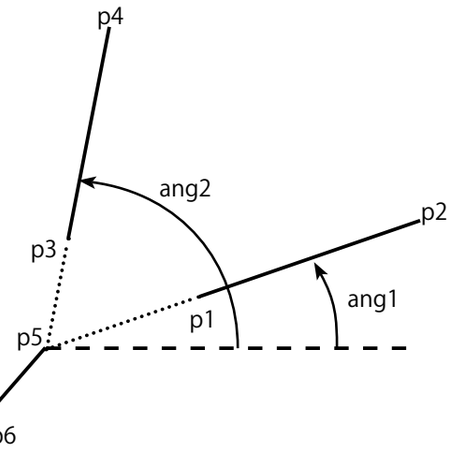
Step6 ー ang1 と ang2 の半分の角度を ang2 に加えた角度が 2 等分線の角度になります。⓲

(setq ang3 (+ (/ (- ang1 ang2)2) ang2))

① ang1 > ang2 の時は線分 (p1-p2) と線分 (p3-p4) の中間に作図する。



ang1 < ang2 の時は①の反対側に作図する。



Step7 ースナップ角度を ang3 に合わせます。（カーソルが ang3 と同じ角度に傾きます。）⓳

(setvar "SNAPANG" ang3)

Step8 ー直交モードにして、線分を作図します。⓴

（上記 Step6 のようにどちらの向きにも作図できます。）
(setvar "ORTHOMODE" 1)

Step9 ー作図が終わると、スナップ角度、直交モード、座標系をワールドに戻します。㉑㉒㉓㉔

(setvar "SNAPANG" 0)
(setvar "ORTHOMODE" 0)
(command "UCS" "W")

AutoLISP の関数

```
(defun C:nitoubun( / ob1 p1 p2 ob2 p3 p4 p5 s ang1 ang2 ang3)
;; 二本の線情報を取得する
(setq ob1 (entsel "¥n 一本目の線を指示 : "))  ;1 本目の線分を選択
(setq p1 (cdr (assoc 10 (entget (car ob1)))))  ; 始点を取得
(setq p1 (trans p1 0 1))  ; ユーザー座標に変換
(setq p2 (cdr (assoc 11 (entget (car ob1)))))  ; 終点を取得
(setq p2 (trans p2 0 1))  ; ユーザー座標に変換

(setq ob2 (entsel " 二本目の線を指示 : "))  ;2 本目の線分を選択
(setq p3 (cdr (assoc 10 (entget (car ob2)))))  ; 始点を取得
(setq p3 (trans p3 0 1))  ; ユーザー座標に変換
(setq p4 (cdr (assoc 11 (entget (car ob2)))))  ; 終点を取得
(setq p4 (trans p4 0 1))  ; ユーザー座標に変換

(setq p5 (inters p1 p2 p3 p4 nil))  ; 交点を求める
;;p5 に近い方を p1 にする
(if (> (distance p5 p1) (distance p5 p2))  ;p5-p1 の方が遠いと
    (setq s p1 p1 p2 p2 s)  ;p1 と p2 を入れ替える
) ;if
;;p5 に近い方を p3 にする
(if (> (distance p5 p3) (distance p5 p4))  ;p5-p3 の方が遠いと
    (setq s p3 p3 p4 p4 s)  ;p3 と p4 を入れ替える
) ;if
```

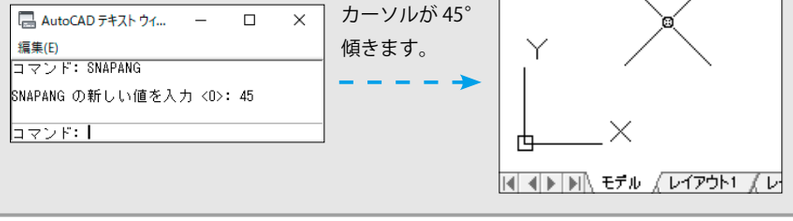
→ Step1（①〜⑤）
→ Step2（⑥〜⑩）
→ Step3（⑪）
→ Step4（⑫〜⑮）

ユーザー座標 (trans 関数) は、P1-111 を参考

Point!

システム変数 [snapang]

[snapang] は現在のビューポートでの、スナップとグリッドの回転角度を現在の UCS を基準に設定します。




```
;p5 から p2 の角度 ang1 と p4 の角度 ang2 を求める
(setq ang1 (angle p5 p2))  ;p5-p2 の角度を求める
(setq ang2 (angle p5 p4))  ;p5-p4 の角度を求める
;ang1 と ang2 の半分の角度が 2 等分線の角度になる
(setq ang3 (+ (/ (- ang1 ang2)2) ang2))  中間の角度
Point!
; カーソルのアングルを変える
(setvar "SNAPANG" ang3)  ; スナップ角度を ang3 に合わせる
(setvar "ORTHOMODE" 1)  直交モードをオンにする
(command "LINE" p5 pause "")  ;p5 から線分を引く
(setvar "SNAPANG" 0)  ; スナップ角度を <0> に戻す
(setvar "ORTHOMODE" 0)  ; 直交モードをオフにする
(command "UCS" "W")  ; ワールド座標系に戻す
) ;end
```

→ Step5（⑯〜⑰）
→ Step6（⑱）
→ Step7（⑲）
→ Step8（⑳〜㉑）
→ Step9（㉒〜㉔）

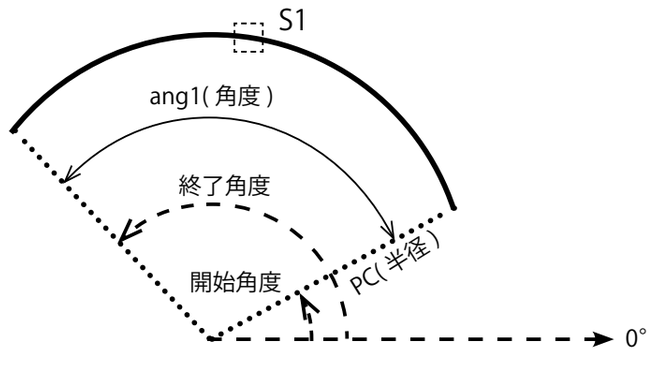
番号	関数名	説　明
①	entsel	選択した 1 つ目の線分の図形名と座標値を取得します。
②	(cdr (assoc 10 (entget (car ob1))))	ob1 から図形名を取りだし、グループ コードが <10> のコード情報 (10 は始点座標) を取得します。
③	trans	既存の図形を使う時は、trans 関数で座標系も変換します。
④	(cdr (assoc 11 (entget (car ob1))))	ob1 から図形名を取りだし、グループ コードが <11> のコード情報 (11 は終点座標) を取得します。
⑤	trans	既存の図形を使う時は、trans 関数で座標系も変換します。
⑥	entsel	選択した 2 つ目の線分の図形名と座標値を取得します。
⑦	(cdr (assoc 10 (entget (car ob2))))	ob2 から図形名を取りだし、グループ コードが <10> のコード情報 (10 は始点座標) を取得します。
⑧	trans	既存の図形を使う時は、trans 関数で座標系も変換します。
⑨	(cdr (assoc 11 (entget (car ob2))))	ob2 から図形名を取りだし、グループ コードが <11> のコード情報 (11 は終点座標) を取得します。
⑩	trans	既存の図形を使う時は、trans 関数で座標系も変換します。
⑪	inters	2 本の線分の交点座標を求めます。
⑫	if	1 本目の線分の交点 p5 に近い方を p1 にします。
⑬	(setq s p1 p1 p2 p2 s)	p2 の方が p5 に近い場合は、p1 と p2 を入れ替えます。
⑭	if	2 本目の線分の交点 p5 に近い方を p3 にします。
⑮	(setq s p3 p3 p4 p4 s)	p4 の方が p5 に近い場合は、p3 と p4 を入れ替えます。
⑯	angle	p5 から p2 への角度をラジアンで求めます。
⑰	angle	p5 から p4 への角度をラジアンで求めます。
⑱	(+ (/ (- ang1 ang2)2) ang2)	2 本の線分の中間の角度を計算します。
⑲	(setvar "SNAPANG" ang3)	カーソルのスナップ角度を <ang3> に合わせます。
⑳	(setvar "ORTHOMODE" 1)	直交モードを <on> にすると、カーソルが <ang3> の角度に固定されます。
㉑	("LINE" p5 pause "")	線分を作図します。始点は p5、終点はユーザーの指示待ち。
㉒	(setvar "SNAPANG" 0)	カーソルのスナップ角度を <0> に戻します。(既定値)
㉓	(setvar "ORTHOMODE" 0)	直交モードを <off> に戻します。(既定値)
㉔	(command "UCS" "W")	UCS をワールド座標系に戻します。(既定値)

inters 関数は、P1-92 を参考

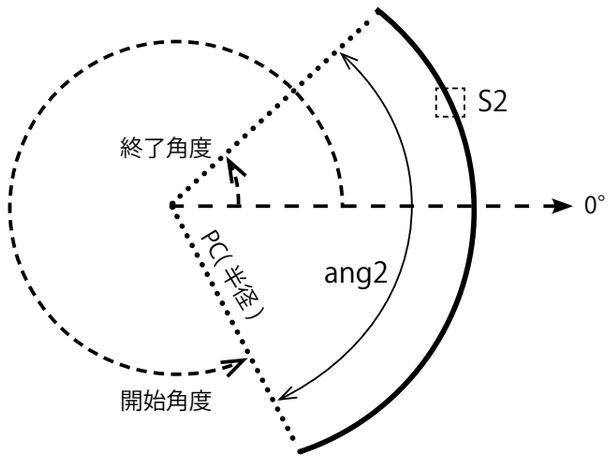
arc_len.lsp	円弧の弧長を測定する
目的：	選択した円弧の弧長を計算します。
主要関数：	<entsel><entget><assoc><cdr>

円弧には始点・終点の概念はなく、開始角度と終了角度の情報があります。つまりどちらから作図しても同じ結果になります。 Point!
従って円弧の角度は、終了角度から開始角度を減算することで求められますが、円弧が角度 0° をまたがっているかどうかで計算方法が違います。
①はまたがっていない場合、②はまたがっている場合です。

①円弧の角度 (ang1) は、終了角度 (pe) ー開始角度 (ps) で計算します。



②０度 < 右水平方向 > をまたがるときは、以下の計算式を使います。
円弧の角度 (ang2) ＝終了角度 pe ＋ (360 ー開始角度 ps)



③円弧の円周は、直径 (2PC) ×π×円弧角度になります。
例えば、半径 100、円弧角度 45° の場合は、2 × 100 ×π (3.14) × (45/360) ≒ 78.5

Step1 ー (setq en1 (entsel "¥n 円弧を指示：")) → 計測する円弧を選択します。❶
(< 図形名 : 7ef039a0> (1316.69 1561.03 0.0)) が取得できます。

Step2 ー (setq en2 (entget(car en1))) → en1 から図形情報を取り出します。❷
以下の情報から、開始角度 (0.722818 ラジアン) と終了角度 (2.32278 ラジアン) が判ります。
また、半径の (446.965) が取得できます。

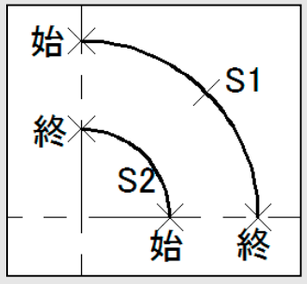
((-1 . < 図形名 : 7ef039a0>) (0 . "ARC") (330 . < 図形名 : 7ef01cf8>) (5 . "2A4") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 .1541.94 1166.91 0.0) (40 . 446.965) (210 0.0 0.0 1.0) (100 . "AcDbArc") (50 . 0.722818) (51 . 2.32278))

Step3 ー円弧の開始角度 (50 . 0.722818) と終了角度 (51 . 2.32278) を取得します。❸❹
図形 <en2> から、開始角度の (50 . 0.722818) を取り出す関数は、assoc 関数です。
そして、開始角度 (50 . 0.722818) は、<50> と <0.722818> のペアですから、2 番目の情報を取得する関数は cdr 関数になります。終了角度も同様です。

円弧のグループ コード	
グループコード	意　味
0	"ARC"
100	サブクラス マーカー (AcDbCircle)
39	厚さ (省略可能、既定 = 0)
10	中心点 (OCS) DXF: X 値、APP: 3D 点
20,30	DXF: 中心点の Y および Z の値 (OCS)
40	半径
100	サブクラス マーカー (AcDbArc)
50	開始角度
51	終了角度
210	押し出し方向 (省略可能、既定 = 0, 0, 1) DXF: X 値、APP: 3D ベクトル
220,230	DXF: 押し出し方向の Y および Z の値 (省略可能)

entsel は、P1-210、entget は、P1-218 を参考

Point!



S1 の円弧は始点 (0 , 100)、終点 (100 , 0)、中心 (0 , 0) で作図しました。
S2 の円弧は始点 (50 , 0)、終点 (0 , 50)、中心 (0 , 0) で作図しました。
entget 関数で取得した 2 つの円弧の情報は以下の通りです。
① S1 の開始角度 (50) と終了角度 (51) は、(50 . 6.28319) (51 . 1.5708) です。
② S2 の開始角度 (50) と終了角度 (51) は、(50 . 6.28319) (51 . 1.5708) です。
以上から、円弧はどちらから始めても開始角度と終了角度は同じであることが判ります。（開始角度から終了角度は反時計回りです。）

AutoLISP の関数

Step4 ー 円弧の半径 (40 . 446.965) を取得します。❺

図形 <en2> から、(40 . 446.965) を取り出す関数は、assoc 関数です。

そして半径 (40 . 446.965) は、<40> と <446.965> のペアですから、2 番目の情報を取得する関数は cdr 関数になります。

Step5 ー 0° (右水平方向) をまたぐかどうかを判断します。❻❼❽

① (if (< pe ps)・・もし pe（終了角度）が ps（開始角度）より小さきときは、
円弧の角度＝終了角度 pe ＋ (360 ー開始角度 ps) で計算します。

② pe（終了角度）が ps（開始角度）より大きいときは、
円弧の角度＝終了角度 (pe) ー開始角度 (ps) で計算します。

Step6 ー ラジアンを度数に変更します。❾

プログラムの前に、ラジアンから度数に変換する関数を記述しておきます。

（関　数）(defun rtd (a) (* 180.0 (/ a pi)))

（計算例）0.785 ラジアンを度数にするには、

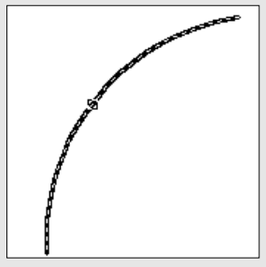
求めたい度数を <a> とすると、180° ： a ＝ π (pi) ： 0.785 になります。

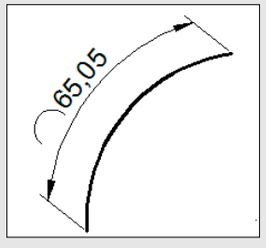
これから <a> の値は、45° になります。

Step7 ー 円弧の長さを計算します。❿

直径 (2PC) ×π×円弧角度から円弧の長さを求めます。

Point!

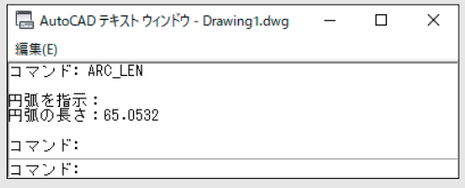




① arc_len.lsp を実行して、円弧を選択します。

②コマンド ラインに円弧の長さが表示されます。

< 円弧の長さ：65.0532>



AutoCAD の寸法コマンドで確認します。

③ [円弧寸法記入] コマンドで、円弧を選択します。

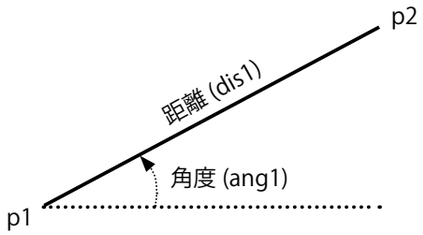
④図面に円弧の長さ寸法が表示されます。

< ⌒65.05>


```
(defun  rtd (a) (* 180.0 (/ a  pi)))  ; ラジアンから度への関数
;rtd 関数は、⑨で使用します。
(defun  C:arc_len ( / en1 en2 ps pe pc ang1 ang2 ad)
❶  (setq  en1  (entsel "¥n 円弧を指示："))  ; 円弧を選択          → Step1
❷  (setq  en2  (entget  (car  en1)))  ; 円弧の情報を取得          → Step2
❸  (setq  ps  (cdr  (assoc  50  en2)))  ; 円弧の開始角を取得      → Step3
❹  (setq  pe  (cdr  (assoc  51  en2)))  ; 円弧の終了角を取得      → Step3
❺  (setq  pc  (cdr  (assoc  40  en2)))  ; 円弧の半径を取得        → Step4
❻  (if  (<  pe  ps)  ; 終了角が開始角より小さければ、1 行目を計算  → Step5
❼    (setq  ang1  (+ pe (- (*  pi  2)  ps)))  ; 小さいとき処理    → Step5
❽    (setq  ang1  (-  pe  ps))  ; 終了角度が開始角度より          → Step5
    ) ;if                       ; 大きいときの処理                → Step5
❾  (setq ang2  (/  360  (rtd  ang1)))  ; ラジアンを度数に変換     → Step6
❿  (setq  ad  (/  (*  (*  2  pc)  pi)  ang2))  ; 弧長の長さを計算 → Step7
⓫  (prompt  "¥n 円弧の長さ：")  ; コマンドラインに表示
⓬  (princ  ad)  ; 計算結果をコマンドラインに表示
⓭  (princ)
) ;end
```

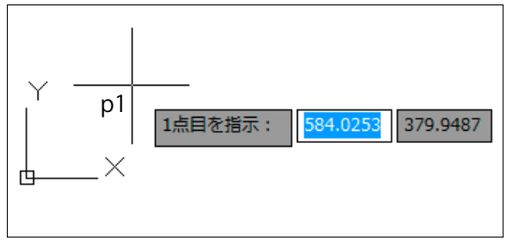
番号	関数名	説　明
①	entsel	取得した円弧の図形名と指示した座標を取得します。
②	entget	en1 の図形名を car 関数で取り出し、その情報を entget で取得します。
③	(cdr (aasoc 50 en2))	始点角度のグループ コードの <50> から、始点角度を取得します。 (cdr (assoc 50 en2)) → (cdr (50 . 0.722818)) → (0.722818)
④	(cdr (assoc 51 en2))	終了角度のグループ コードの <51> から、始点角度を取得します。 (cdr (assoc 51 en2)) → (cdr (51 . 2.32278)) → (2.32278)
⑤	(cdr (assoc 40 en2))	円弧半径のグループ コードの <40> から、円弧の半径を取得します。 (cdr (assoc 40 en2)) → (cdr (40 . 466.965)) → (466.965)
⑥	(if (< pe ps)	0° をまたぐかどうかの判断（終了角度が始点角度より小さいか大きいかで処理を分岐します。）
⑦	(+ pe (- (* pi 2) ps))	終了角度の方が小さければ <0° > をまたいだ計算を行います。 （360° から開始角度を引いた角度を終了角度に加えます。）
⑧	(- pe ps)	またがないときの処理を記述します。
⑨	(/ 360 (rtd ang1))	円弧の角度をラジアンから度数に変換して計算します。
⑩	(/ (* (* 2 ec) pi) ang2)	円周＝直径×π×(中心角を 360° で割った値)
⑪	prompt	コマンドラインに、" 円弧の長さ：" を表示します。
⑫	(princ ad)	計算結果の円弧の長さを表示します。
⑬	princ	最後の行に、<nil> を表示させません。

Polar_1.lsp	線分を作成する
目的：	始点と角度と長さを指定して線分を作図します。
主要関数：	<getpoint><getangle><getdist><polar>



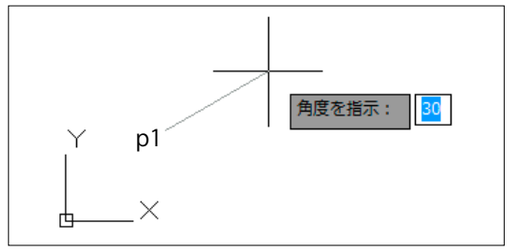
Step1 ーマウスで、1 点目 (p1) を指示します。<getpoint> ❶

① 1 点目を指示：



Step2 ーキーボードから角度を入力します。<getangle> ❷

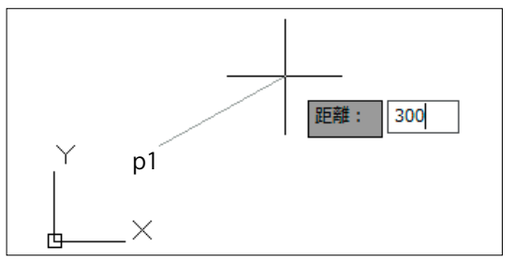
②角度を指示：



角度をキーボードで数値入力するか、またはマウスで画面上を指示します。

Step3 ーキーボードから距離を入力します。<getdist> ❸

③距離：



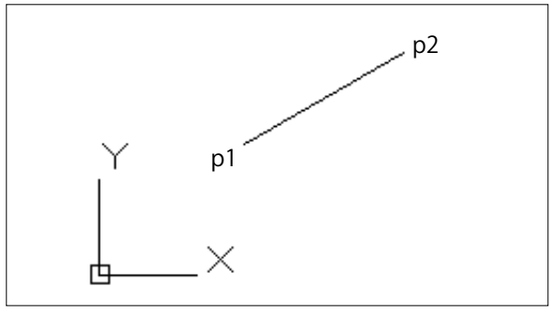
距離をキーボードで数値入力するか、またはマウスで画面上を指示します。

Step4 ー polar 関数を使い、角度と距離から p2 の座標を計算します。<polar> ❹

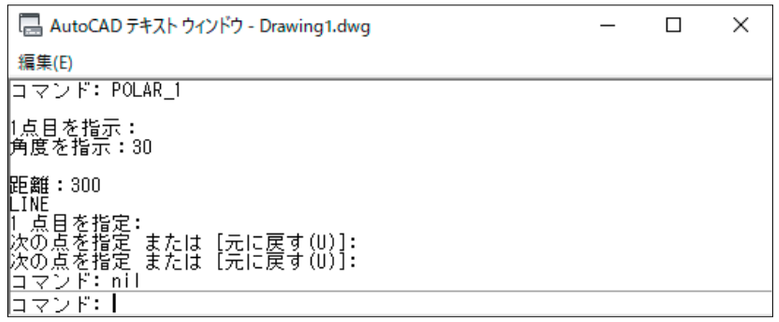
polar 関数の書式は、(polar　基点　角度 < ラジアン >　距離) です。

Step5 ー "LINE" コマンドで、p1 から p2 までの線分を作図します。<LINE> ❺

Step5 ー P1-P2 の線分を作図




```
(defun C:Polar_1( / p1 ang1 dis1 p2)
❶ (setq p1  (getpoint "¥n1 点目を指示：" ))  ;1 点目を指示          → Step1
❷ (setq ang1 (getangle p1 "¥n 角度を指示：" ))  ; 角度を入力        → Step2
❸ (setq dis1 (getdist p1 "¥n 距離：" ))  ; 距離を入力               → Step3
;p1、ang1、dis1 から p2 の座標を求める
❹ (setq p2  (polar p1 ang1 dis1))                                  → Step4
;p1-p2 に線分を作図
❺ (command "LINE" p1 p2 "" )                                       → Step5
) ;end
```



番号	関数名	説　明
①	getpoint	ユーザーがマウスで指示した座標を変数 p1 にセットします。
②	getangle	ユーザーがマウスかキーボードで入力した数値を変数 ang1 にセットします。
③	getdist	ユーザーがマウスかキーボードで入力した数値を変数 dis1 にセットします。
④	polar	polar 関数で計算した座標を変数 p2 にセットします。
⑤	command	線分コマンドで、点 p1 から点 p2 まで線分を作図します。

☞getpoint は P1-156、getangle は P1-155、getdist は P1-156 を参考

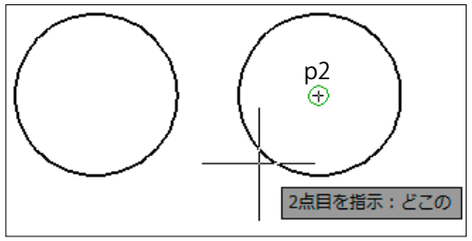
Polar_2.lsp	2 点の中点から線分を作成する
目的：	2 点を指示して、その中点から線分を作図します。
主要関数：	<getpoint><angle><distance><polar>

Step1 ーマウスで、1 点目 (p1) を指示します。（左の円の中心を指示します。）❶

1 点目を指示：

Step2 ーマウスで、 2 点目 (p2) を指示します。（右の円の中心を指示します。）❷

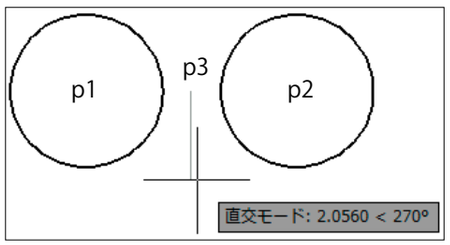
2 点目を指示：



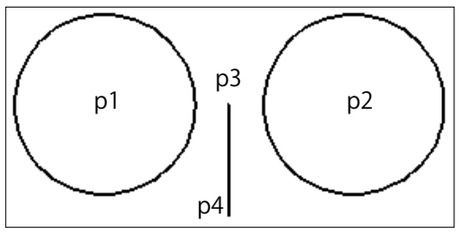
2 点間の中点までの角度 (ang1) と距離 (dis1) はプログラムが計算します。
角度は (angle p1 p2)
→ ラジアンで取得
距離は (/ (distance p1 p2) 2)
→ p1-p2 の距離の半分が中点までの距離

Step3 ー polar 関数を使い、p1 と p2 の中点座標 (p3) を計算します。❸❹❺
2 点の中点 p3 から線分を開始します。❻

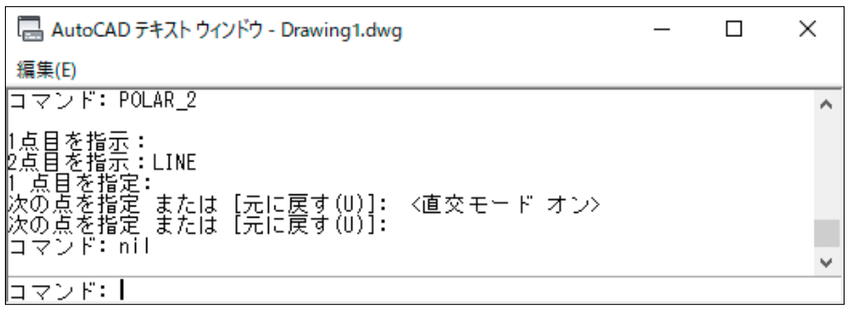
中点座標を計算し、始点 (p3) にセット



線分の終点 (p4) を指示



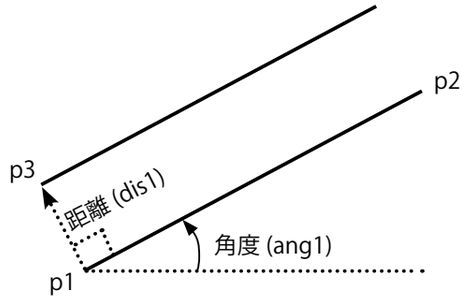

```
(defun C:Polar_2( / p1 p2 ang1 dis1 p3)
❶  (setq p1  (getpoint "¥n1 点目を指示：" ))  ;1 点目を取得          → Step1
❷  (setq p2  (getpoint "¥n2 点目を指示：" ))  ;2 点目を取得          → Step2
❸  (setq ang1 (angle p1 p2))  ;p1-p2 の角度を ang1 にセット
❹  (setq dis1 (/ (distance p1 p2) 2))  ;p1-p2 の距離の半分をセット
❺  (setq p3  (polar p1 ang1 dis1))  ;p1 から角度 ang1、距離 dis1 の距離が p3   → Step3
❻  (command "LINE" p3 pause "")  ; 線分を作成
) ;end
```



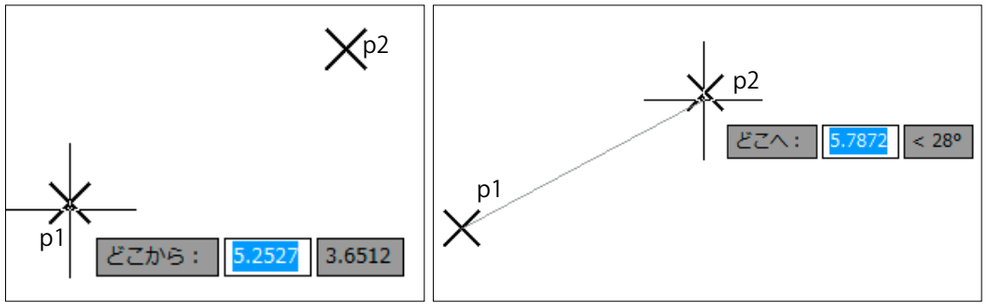
番号	関数名	説　明
①	getpoint	マウスで指示した座標を変数 p1 にセットします。
②	getpoint	マウスで指示した座標を変数 p2 にセットします。
③	angle	2 点間の角度の数値を変数 ang1 にセットします。 書式は、(angle 1 点目 2 点目）です。
④	distance	2 点間の距離を取得する関数。p1 と p2 の距離を 2 で割って、中点までの距離を出しています。 書式は、(distance 1 点目 2 点目）です。
⑤	polar	polar 関数で計算した座標を変数 p3 にセットします。 書式は、(polar 基点 角度 < ラジアン > 距離）です。
⑥	command	線分コマンドを使用して、中点 p3 から線分を出しています。 始点の P3 から始まった後、pause でユーザーの入力待ち状態で止めています。 2 点目を指示すると、<""> で線分コマンドを終了させます。

getpoint は P1-156 を参考

para1.lsp	2 点に平行な線分を作成する
目的：	マウスで指示した 2 点に平行な線分を作成します。
主要関数：	<getpoint><angle><distance><polar>



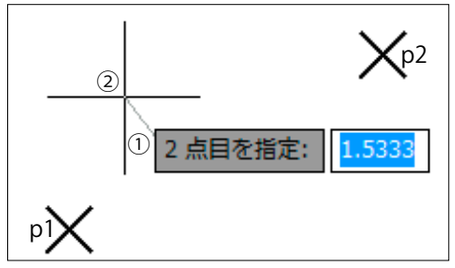
Step1 ーマウスで、1 点目 (p1) と 2 点目 (p2) を指示します。❶❷



指示した 2 点から、p1 と p2 の角度 (ang1) を計算します。❸

→ ang1 (angle p1 p2)

Step2 ー平行線の幅 (dis1) の入力を促します。❹



コマンド : (setq dis1 (getdist "¥n 幅を入力："))

幅を入力：2 点目を指定 :（①から②）

→ マウスで 1 点目を指示すると、<2 点目を指定 :> のメッセージが表示されます。

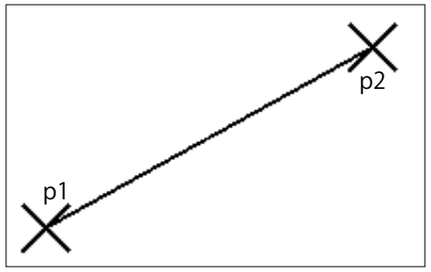
（キーボードからでも入力できます。）

POLAR 関数を使い、p1 から 90° の角度を計算します。(p3) ❺

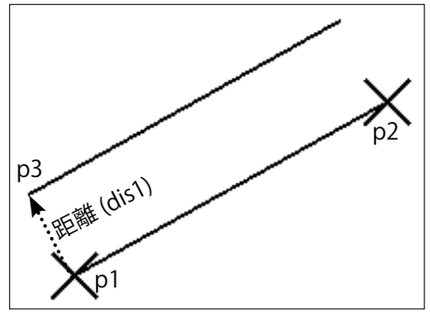
ang1 から 90° の方向は、(+ ang1 (* pi 0.5)) になります。<90° → 0.5pi、180° → pi>

（進行方向 <p1-p2> の左側に平行線を作図します。）

Step3 ー p1 と p2 を結ぶ線分を作成します。❻



Step4 ー作成した線分を、複写コマンドを使い、p3 の位置に複写します。❼

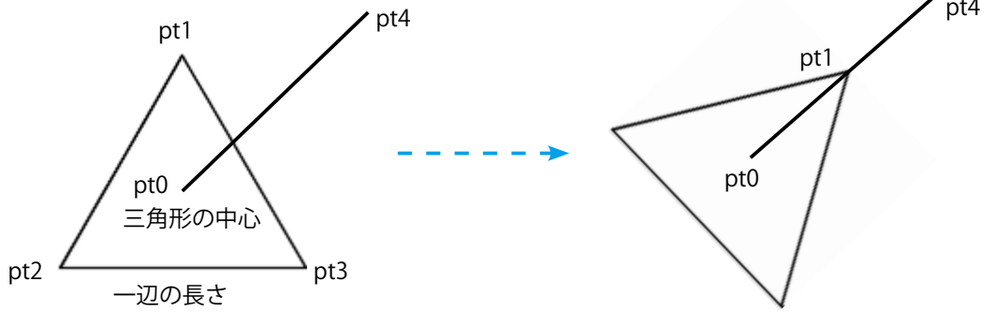



```
(defun C:para1( / p1 p2  p3 ang1 dis1)
❶   (setq  p1  (getpoint  "¥n どこから："))  ; 線分の始点を取得
❷   (setq  p2  (getpoint  p1  "¥n どこへ："))  ; 線分の終点を取得          → Step1
❸   (setq  ang1 (angle  p1  p2))  ; 始点と終点の角度を計算
❹   (setq  dis1   (getdist  "¥n 幅を入力："))  ; オフセットの幅を入力      → Step2
❺   (setq  p3  (polar  p1  (+  ang1  (*  pi  0.5))  dis1))  ;p3 の位置を計算
❻   (command  "LINE"  p1  p2  "")  ; 線分を作図                           → Step3
❼   (command  "COPY"  "L"  ""  p1  p3)  ; 線分を複写                      → Step4
) ;end
```

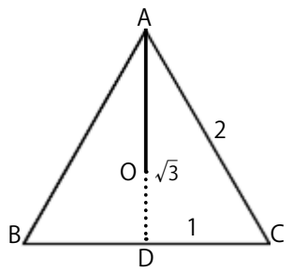
番号	関数名	説　明
①	getpoint	マウスで指示した座標を変数 p1 にセットします。
②	(getpoint p1)	getpoint の後に p1 を付けると、p1 からのラバーバンドが表示されます。
③	angle	2 点間の角度の数値を変数 ang1 にセットします。 書式は、(angle 1 点目 2 点目）です。
④	getdist	距離の数値を変数 dis1 にセットします。
⑤	polar	polar 関数で計算した座標を変数 p3 にセットします。 書式は、(polar 基点 角度 < ラジアン > 距離）です。
⑥	("LINE" p1 p2 "")	p1 から p2 に線分を作図します。
⑦	("COPY" "L" "")	最後の図形 <Last>（線分）を p1 を基点として、p3 の位置に複写します。

☞getpoint、getdist は P1-156 を参考

ang_3hen.lsp	1 辺の長さと中心を指定して正三角形を作図し、自由な位置に回転する
目的：	線分の端点 (pt0) を中心に正三角形を作図して、頂点を線分の端点 (pt4) 方向に合わせます。
主要関数：	<getpoint><getdist><polar><entlast><ROTATE>



Step1 ー三角形の一辺の長さ <（例）100> を入力し、三角形の中心を指示します。❶❷❸



①直角三角形の辺の長さの比率は
CD : AC : AD = 1 : 2 : √3 ですから、
CD が <50> であれば、AD の長さは、50 √3になります。

②正三角形の中心 <O> までの距離は
AO : OD = 2 : 1 ですから、
AO の長さは、(50 √3 × 2/3) になります。

Step2 ー√3 は LISP 式では (sqrt 3) です。❹
従って、AO の長さ (50 √3 × 2/3) は、(/ (* (* 50 (sqrt 3)) 2) 3) となります。
頂点 A は中心 O から角度 90° (pi / 2)、長さは (/ (* (* 50 (sqrt 3)) 2) 3) の位置になります。

Step3 ー頂点 B は、頂点 A から角度 240° (4/3 × pi)、長さは 100 の位置になります。❺
(4/3 × pi) は LISP 式では、(/ (* 4 pi) 3) になります。

Step4 ー頂点 C は、頂点 B から角度 0° 、長さは 100 の位置になります。❻

Step5 ー頂点 ABC をポリラインで作図します。❼

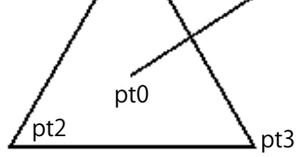
Step6 ーこの三角形は最後に作図した図形ですから、この図形を取得する関数は、entlast です。❽

Step7 ー回転コマンドを使用するとき、オプション "R" を使ってマウスで回転の方向を指示できます。
（参照する角度は点 <pt0-pt1>、新しい角度は点 <pt0-pt4> です。）❾

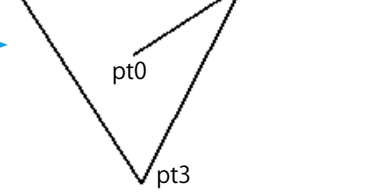

```
(defun c:ang_3hen()
❶ (setq pt0 (getpoint "¥n 三角形の中心を指示：")) ; 中心の座標を取得
❷ (setq dist1 (getdist pt0 "¥n 一辺の長さは？ ")) ; 三角形の 1 辺の長さを取得
❸ (setq dist2 (/ dist1 2)) ;1 辺の長さの半分を計算（三角形の重心を求めるため）
  ; 三角形の重心を基点に各頂点の座標を計算する
❹ (setq pt1 (polar pt0 (/ pi 2) ( / ( * ( * dist2 (sqrt 3)) 2 ) 3))) ;pt1 の座標
❺ (setq pt2 (polar pt1 ( / ( * 4 pi) 3) dist1)) ;pt2 の座標
❻ (setq pt3 (polar pt2 0 dist1)) ;pt3 の座標
  ; 計算した各頂点にポリラインを作図
❼ (command "PLINE" pt1 pt2 pt3 "C")
❽ (setq obj1 (entlast)) ; 三角形を回転するために選択
❾ (command "ROTATE" obj1 "" pt0 "R" pt0 pt1 pause) ; 三角形を回転する
(princ)
);end
```

Step1
Step2
Step3
Step4
Step5
Step6
Step7

正三角形を作図します。



マウスで回転方向 (pt4) を指示します。



番号	関数名	説　明
①	getpoint	マウスで指示した座標を取得して、変数 <pt0> にセットします。
②	getdist	キーボードか画面上からマウスで距離の入力を受け付けます。
③	(/ dist1 2)	三角形の 1 辺の長さの半分を計算します。前ページの CD の距離です。
④	polar	pt0 からの方向と距離で、pt1 の座標を計算します。
⑤	polar	pt1 からの方向と距離で、pt2 の座標を計算します。
⑥	polar	pt2 からの方向と距離で、pt3 の座標を計算します。
⑦	PLINE	ポリラインで三角形を作図します。"C" でクローズします。
⑧	entlast	最後に作成したオブジェクトを取得する関数です。
⑨	ROTATE	回転コマンドの参照 <R> を使って、マウスで回転方向を指示します。<pause> でユーザーが点 pt4 を指示するのを待ちます。

☞ getpoint、getdist は P1-156 を参考
entlast は、P1-209 を参考

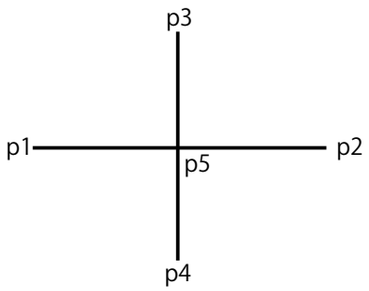
3 座標計算のジオメトリック関数（線分の交点計算）

2 次元空間での交点計算

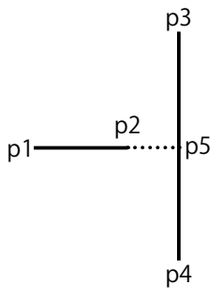
INTERS	2 本の線分の交点を求めます。(作図では 5 番目の引数に <nil> を付けるのが安全)

線分 A(p1,p2) と線分 B(p3,p4) の交点 <p5> を求めます。
①② < 線分 A と線分 B が交差している場合 >　　③ < 線分 A と線分 B が交差していない場合 >

①と②



③



①線分同士が交わる場合

```
(setq p5 (inters p1 p2 p3 p4))
```

②線分同士が必ず交わる場合

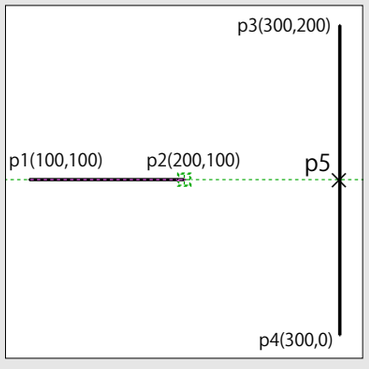
```
(setq p5 (inters p1 p2 p3 p4 T))
```

③線分同士が交わらない場合（交わっている場合でも有効）

```
(setq p5 (inters p1 p2 p3 p4 nil))
```

2 本の線分が交差しているか、していないかの検証にも使います。
(setq pt5 (inters p1 p2 p3 p4))
→ もし、pt5 の値が <nil> であれば、この 2 本の線分は交差していないことになります。

Point!

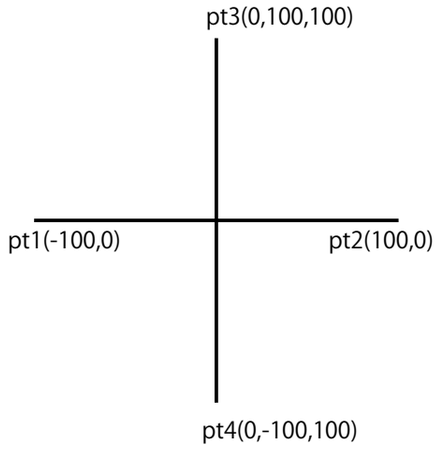



```
(setq p1 '(100 100))            ;p1 の座標をセット
(setq p2 '(200 100))            ;p2 の座標をセット
(setq p3 '(300 200))            ;p3 の座標をセット
(setq p4 '(300 0))              ;p4 の座標をセット
(command "LINE" p1 p2 "")       ;p1-p2 の線分を作図
(command "LINE" p3 p4 "")       ;p3-p4 の線分を作図

;2 本の線分から、p5 の座標を計算します。
(setq p5 (inters p1 p2 p3 p4 nil))
→  p5 = (300.0 100.0) が返ります。
```

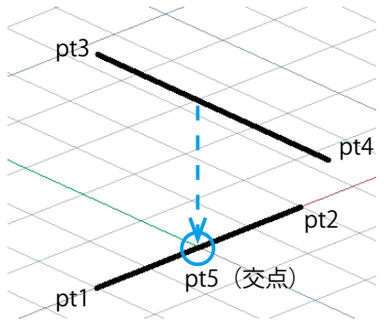

3 次元空間での交点計算

下図の 2 本の線分は、pt1(-100,0) と pt2(100,0) の線分と pt3(0,100,100) と pt4(0,-100,100) の線分です。
線分 pt1-pt2 は Z=0 の平面上にあり、線分 pt3-pt4 は Z=100 の位置にありますから、この 2 本の線分が交差することはありません。
しかし、pt1 と pt2 の座標値を X と Y だけで指定した場合は、inters 関数で交点を求めることができます。点のいずれかが 2D の場合、inters 関数は線分を現在の作図平面に投影して、2D の交点のみを検出します。
4 点の全てが 3D 座標 (X,Y,Z を持つ) の場合は、この 2 本の線分は交点を持たず、nil が返ります。

真上から見た図



斜め上から見た図




```
(setq pt1 (list -100 0))                    → 点 pt1 に (-100  0) をセット
(setq pt2 (list 100 0))                     → 点 pt2 に (100   0) をセット
(setq pt3 (list  0 100 100))                → 点 pt3 に (0  100  100) をセット
(setq pt4 (list 0  -100  100))              → 点 pt4 に (0  -100  100) をセット

(command "LINE" pt1 pt2 "")                 → 点 pt1 から点 pt2 へ線分を作図
(command "LINE" pt3 pt4 "")                 → 点 pt3 から点 pt4 へ線分を作図
(setq pt5 (inters pt1 pt2 pt3 pt4 nil))     → 2 本の線分の交点を求める

コマンドラインから、!pt5 と入力します。      → (0.0 0.0) が返ります。
```

すべての点は、現在の UCS で表されます。4 点の全てが 3D 座標 の場合、inters 関数は 3D 座標の交点をチェックします。
もし、pt1 の座標が (-100,0,0) であり、pt2 の座標が (100,0,0) であれば、この 2 本の交点は検出されません。

第 5 節　データ変換関数

AutoCAD のシステム変数に [CDATE] があります。これは、現在の日時を 10 進数形式で表示する関数です。

①キーボードから次のように入力します。
　(setq date1 (getvar "CDATE"))　　→ <2.02007e+07> が返ります。
②これだけでは判りづらいので文字列に変換します。
　(setq date1 (rtos (getvar "CDATE")))　　→ <"20200720.1516"> が返ります。
③最初から 4 つ目までが西暦の年になります。
　(substr date1 1 4)　　→ <"2020"> が返ります。
④ 5 番目から 2 つは月になります。
　(substr date1 5 2)　　→ <"07"> が返ります。
⑤ 7 番目から 2 つは日になります。
　(substr date1 7 2)　　→ <"20"> が返ります。
⑥ 10 番目から 2 つは時になります。（9 番目は小数点 <.> のため無視します。）
　(substr date1 10 2)　　→ <"15"> が返ります。
⑦ 12 番目から 2 つは分になります。
　(substr date1 12 2)　　→ <"16"> が返ります。

このように [CDATE] だけの情報 <2.02007e+07> では何のことか判りませんが、数値情報を文字情報に変換することで本日の日時情報であることが理解できます。
この節では、整数や実数、文字列情報などをお互いに変換しあう関数を学びます。

使用頻度の高い AutoLISP の変換関数を、次の表に示します。（順不同）

①実数 ↔ 整数への変換関数（REAL ↔ INT）	
関　数	説　明
(float number)	数値を実数に変換して返します。
(fix number)	実数の小数点以下を切り捨てて整数に変換して返します。

②文字 → 数値への変換関数（STR → INT、REAL）	
関　数	説　明
(atoi string)	文字列を整数に変換して返します。
(atof string)	文字列を実数に変換して返します。
(angtof string [mode])	角度を表す文字列をラジアン単位の実数 (浮動小数点) 値に変換して返します。
(distof string [mode])	実数（浮動小数点）を表す文字列を、実数値に変換して返します。

③数値 → 文字への変換関数（INT、REAL → STR）	
関　数	説　明
(itoa int)	整数を文字列に変換して返します。
(rtos number [mode [precision]])	実数を文字列に変換して返します。
(angtos angle [mode [precision]])	ラジアン単位の角度の値を文字列に変換して返します。

④計測単位同士の変換関数　&　座標系同士の変換関数	
関　数	説　明
(trans pt from to [disp])	ある座標系から別の座標系に、点 (または変位) を変換します。
(cvunit value from to)	ある計測単位から別の計測単位に値を変換します。

1　実数 ↔ 整数への変換関数（REAL ↔ INT）

FLOAT	数値を実数に変換します。
(float 3.14)	→ 3.14
(float 3)	→ 3.0
FIX	**数値を整数に変換します。**
(fix 3.14)	→ 3
(fix 3)	→ 3

切り捨て・切り上げ・四捨五入

①円周率 (3.14159・・・) の小数点 3 桁以下を切り捨てる処理。

(setq a (* pi 100))　　→ 314.159・・・（円周率を 100 倍します。）
(setq a (float (fix a)))　　→ 314.0（fix で整数止めにした後、実数に変換）
(setq a (/ a 100))　　→ 3.14（314.0 を 100 で割って、元に戻します。）
!a　　→ コマンドラインに 3.14 が表示されます。

②円周率 (3.14159・・・) の小数点 3 桁目を切り上げて、小数点 2 桁にする処理。

(setq a (* pi 100))　　→ 314.159・・・（円周率を 100 倍します。）
(setq a (+ a 0.9))　　→ 315.059・・（小数点 1 桁に、0.9 を加えます。）
(setq a (float (fix a)))　　→ 315.0（fix で整数止めにした後、実数に変換）
(setq a (/ a 100))　　→ 3.15（315.0 を 100 で割って、元に戻します。）
!a　　→ コマンドラインに 3.15 が表示されます。

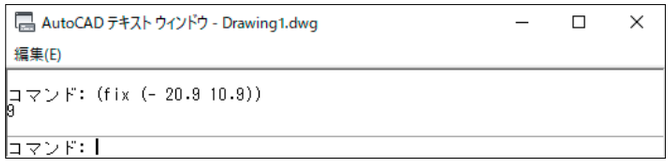
③円周率 (3.14159・・・) の小数点 3 桁目を四捨五入する処理。

(setq a (* pi 100))　　→ 314.159・・・（円周率を 100 倍します。）
(setq a (+ a 0.5))　　→ 314.659・・（小数点 1 桁に、0.5 を加えます。）
(setq a (float (fix a)))　　→ 314.0（fix で整数止めにした後、実数に変換）
(setq a (/ a 100))　　→ 3.14（314.0 を 100 で割って、元に戻します。）
!a　　→ コマンドラインに 3.14 が表示されます。

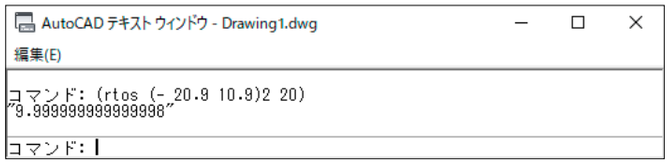
fix による計算誤差

①前ページでは、π (円周率) を整数に丸めるときに fix 関数を使いました。
(fix pi)　→ <3> が返ります。

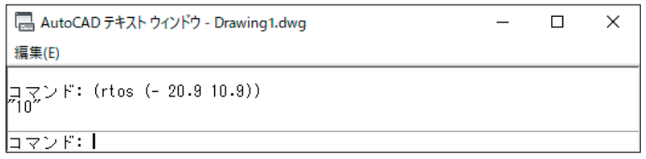
②減算する関数を使った時に、以下のようになる場合があります。
(fix (- 20.9 10.9))　→ <9> が返ります。(<10> ではありません。)



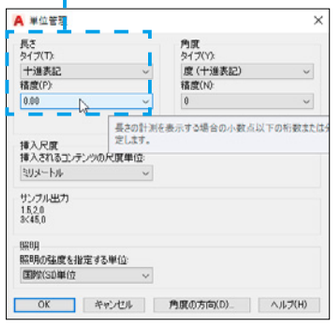
③実数を文字列に変換する関数 [rtos] で確認してみると、以下のような循環小数になっています。
コマンド : (rtos (- 20.9 10.9)2 20)　→　"9.999999999999998"（十進法で小数点以下 20 桁まで表示）



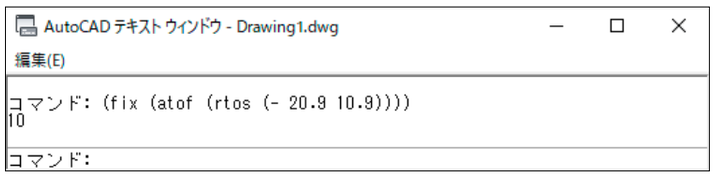
④システム変数 [LUPREC]（長さの単位の表示精度を設定）を <2> で再度計算してみます。
(rtos (- 20.9 10.9))　→ <"10"> が返ります。



システム変数 [LUPREC]



⑤このようにシステム変数 [LUPREC] の表示精度によって、返り値が違う場合が起こります。
こうした誤差を回避するために、一度文字列に変換してから fix 関数を使うのが無難です。
(fix (atof (rtos (- 20.9 10.9))))　→ <10>
こうすると、システム変数 [LUPREC] で丸めて、整数部だけが正確に返ります。



2 文字 → 数値への変換関数（STR → REAL、INT）

ATOF	文字列を実数に変換します。

(atof "3.14")　→ 3.14
(atof "3")　→ 3.0

ATOI	文字列を整数に変換します。

(atoi "3.14")　→ 3
(atoi "3")　→ 3

DISTOF	実数（浮動小数点）を表す文字列を、実数値に変換します。

(setq txt1 "3.14")　→ 変数 text1 に、"3.14" を代入。
(setq txt1 (distos txt1 2 1))　→ txt1 を十進表記で小数点以下 1 桁表示。
コマンド : !txt1　→ コマンドラインに !txt1 と入力します。
3.1　→ コマンドラインに <3.1> が表示されます。

rtos の表示形式コード

モード	表示形式
1	指数表記
2	十進表記
3	工業図面表記
4	建築図面表記
5	分数表記

< モード > を省略した場合、LUNITS の現在値を使用します。

ANGTOF	角度を表す文字列を角度の実数 (浮動小数点) 値に変換します。

(setq ang1 (getstring "¥n 角度を入力 : "))　→ "45" を入力。
(setq ang1 (angtof ang1))　→ ang1 をラジアンに変換します。
コマンド : !ang1　→ コマンドラインに !ang1 と入力します。
0.785398　→ コマンドラインに <0.785398> が表示されます。

angtof の表示形式コード

モード	表示形式
0	度（十進数）
1	度 / 分 / 秒
2	グラジェント
3	ラジアン
4	測量単位

< モード > を省略した場合、AUNITS の現在値を使用します。

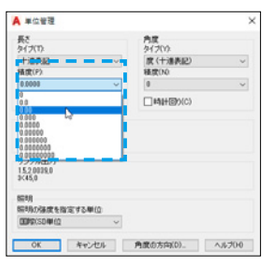
distof 関数	長さと距離の文字列を実数に変換する
書式	(distof 変換する文字列 モード)

distof 関数 (距離・長さの文字列を実数に変換) は、rtos 関数と反対の働きをします。
結果の文字列の形式は、AutoCAD のシステム変数の値によってコントロールできます。
モードは、[LUNITS] に従って設定し、小数点以下の表示は [LUPREC] で指定します。

変換式	モード	結果
(distof "5.2500E+01" 1)	1 = 指数表記	52.5
(distof "52.5" 2)	2 = 十進表記	52.5
(distof "4'-4.5¥"" 3)	3 = 工業図面表記	52.5
(distof "4'-4 1/2¥"" 4)	4 = 建築図面表記	52.5
(distof "52 1/2" 5)	5 = 分数表記	52.5

LUNITS（長さの単位）

LUPREC（長さの表示精度）



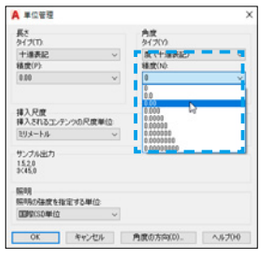
angtof 関数	角度を表す文字列を実数に変換する
書式	(angtof 変換する文字列 モード)

角度を表す文字列をラジアン単位の実数に変換します。ANGTOF 関数は ANGTOS 関数の反対の働きをします。
結果の文字列の形式は、AutoCAD のシステム変数の値によってコントロールできます。
モードは、[AUNITS] に従って設定し、小数点以下の表示は [AUPREC] で指定します。

(setq a "30.0000") の時		
変換式	モード	結果
(setq b (angtof a 0))	0 = 十進数（度）	b = 0.523599
(setq b (angtof a 1))	1 = 度 / 分 / 秒	b = 0.523599
(setq b (angtof a 2))	2 = グラジェント	b = 0.471239
(setq b (angtof a 3))	3 = ラジアン	b = 4.86726
(setq b (angtof a 4))	4 = 測量単位	b = 0.523599

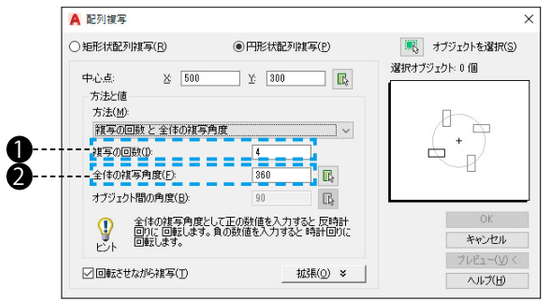
AUITS（角度の単位）

AUPREC（角度の表示精度）



AutoCAD のダイアログ ボックスの数値文字

下図は AutoCAD の [ARRAYCLASSIC] コマンドのダイアログ ボックスです。



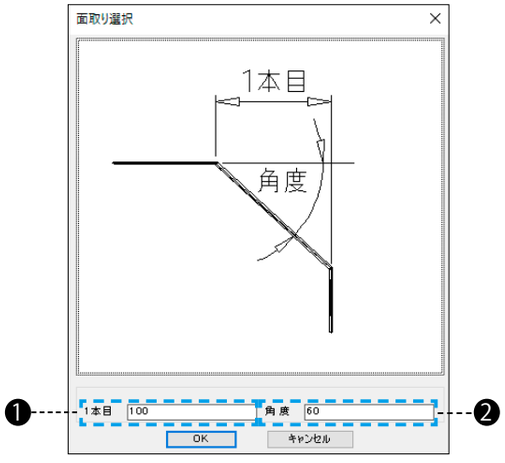
①の [複写の回数] と②の [全体の複写角度] は数値の入力ですが、ダイアログ ボックスの値は文字情報になっています。
つまり、①の値は <4> ではなく <"4"> です。
また②の値は <360> ではなく <"360"> です。

この数値を AutoCAD 側で数値に変換しています。

コマンド : _arraypolar
オブジェクトを選択 : 認識された数 : 1
オブジェクトを選択 :
種類 = 円形状 自動調整 = はい
配列複写の中心を指定 または [基点 (B)/ 回転軸 (A)]:
項目数を入力 または [項目間の角度 (A)/ 式 (E)] <4>: ①← (atoi 関数で整数に変換)
全体の複写角度を入力 または [式 (EX)] <360>: ②← (angtof 関数で角度に変換)

ユーザー作成のダイアログ ボックスの数値文字

下図はユーザー作成の [S_Edit21.lsp] のダイアログ ボックスです。



S_Edit2.lsp は、第 2 部 3 章 5 節 (P2-99) を参考

①の [1 本目] と②の [角度] は数値の入力ですが、ダイアログ ボックスの値は文字情報になっています。
つまり、①の値は <100> ではなく <"100"> です。また②の値は <60> ではなく <"60"> です。

この数値を AutoLISP 側で数値に変換しています。

```
(setvar "CHAMFERC" (atof sen1)) ①← (atof 関数で実数に変換)
(setvar "CHAMFERD" (angtof kaku 0)) ②← (angtof 関数で角度に変換)
(prompt "¥n1 本目の線分と 2 本目の線分を選択：")
(command "CHAMFER")
```

3 数値 → 文字への変換関数（REAL、INT → STR）

ITOA	整数を文字列に変換します。

(itoa 3) → "3"
(itoa -3) → "-3"

RTOS	実数を文字列に変換します。（rtos 実数 モード 小数以下の桁数）

(setq txt1 3.14) → 変数 text1 に、3.14 を代入。
(setq txt1 (rtos txt1 2 1)) → txt1 を十進表記で小数点以下 1 桁表示。
コマンド : !txt1 → コマンドラインに !txt1 と入力します。
"3.1" → コマンドラインに "3.1" が表示されます。

rtos の表示形式コード	
モード	表示形式
1	指数表記
2	十進表記
3	工業図面表記
4	建築図面表記
5	分数表記

< モード > を省略した場合、LUNITS の現在値を使用します。
< 小数点以下の桁数 > を省略すると、AutoCAD のシステム変数 LUPREC の現在値を使用します。

ANGTOS	角度を文字列に変換します。（angtos 角度 モード 小数点以下の桁数）

(setq ang1 (getangle "¥n 角度を入力：")) → 45 を入力（0.785398< ラジアン >）。
(setq ang1 (angtos ang1 0 0)) → ang1 を角度に変換（小数点以下なし）。
コマンド : !ang1 → コマンドラインに !ang1 と入力します。
"45" → コマンドラインに "45" が表示されます。

angtos の表示形式コード	
モード	表示形式
0	度（十進数）
1	度 / 分 / 秒
2	グラジェント
3	ラジアン
4	測量単位

< モード > を省略した場合、angtos 関数は AutoCAD のシステム変数 AUNITS の現在値を使用します。
< 小数点以下の桁数 > を省略すると、AutoCAD のシステム変数 AUPREC の現在値を使用します。

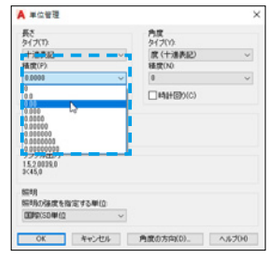
rtos 関数	実数（長さ・距離）を文字列に変換する
書式	(rtos 数値 モード 表示精度)

rtos 関数 (実数を文字列に変換) は、AutoCAD で使用される数値を、出力で使用できる文字列値に、または文字データに変換します。RTOS 関数は DISTOF 関数の反対の働きをします。
結果の文字列の形式は、AutoCAD のシステム変数の値によってコントロールされます。単位と精度は、LUNITS と LUPREC の設定 に従います。（表示精度を省略した場合は、現在の設定が適用）

(setq a 52.5) の時		
変換式	モード	結果
(setq b (rtos a 1 4))	1 = 指数表記	b = "5.2500E+01"
(setq b (rtos a 2 2))	2 = 十進表記	b = "52.5"
(setq b (rtos a 3 2))	3 = 工業図面表記	b = "4'-4.5¥""
(setq b (rtos a 4 2))	4 = 建築図面表記	b = "4'-4 1/2¥""
(setq b (rtos a 5 2))	5 = 分数表記	b = "52 1/2"

LUNITS（長さの単位）

LUPREC（長さの表示精度）



angtos 関数	ラジアン単位の角度を文字列に変換する
書式	(angtos 数値 モード 表示精度)

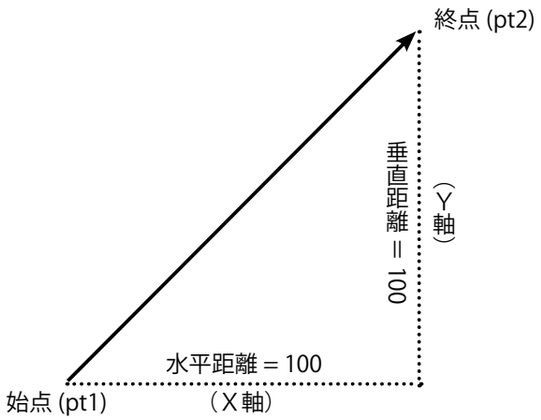
angtos 関数 (角度を文字列に変換) は、AutoCAD で使用される数値を、出力で使用できる文字列値に、または文字データに変換します。ANGTOS 関数は ANGTOF 関数の反対の働きをします。
結果の文字列の形式は、AutoCAD のシステム変数の値によってコントロールされます。単位と精度は、AUNITS と AUPREC の設定 に従います。（表示精度を省略した場合は、現在の設定が適用）

(setq a 3.14159) の時		
変換式	モード	結果
(setq b (angtos a 0 0))	0 = 度（十進数）	b = "180"
(setq b (angtos a 1 4))	1 = 度 / 分 / 秒	b = "179d59'59¥""
(setq b (angtos a 2 4))	2 = グラジェント	b = "199.9998g"
(setq b (angtos a 3 4))	3 = ラジアン	b = "3.1416r"
(setq b (angtos a 4 0))	4 = 測量単位	b = "N 90d W"

AUITS（角度の単位）

AUPREC（角度の表示精度）

rtos 関数	実数を文字列に変換
相対座標入力で線分コマンド (LINE) を使う	



線分コマンドの時、始点をマウスで指示をして、終点の位置をキーボードから相対座標入力する場合は、(@X の増加分 ,Y の増加分) の形になります。

LISP のプログラムでは、次のようにユーザーに X と Y の値を要求します。

```
(setq  dist_x (getreal  "¥nX の増加分："))          → ユーザーが <100> と入力。
(setq  dist_y (getreal  "¥nY の増加分："))          → ユーザーが <100> と入力。
```

キーボードからの入力では、(@100,100) ですから、この形に文字を連結します。（要素は 4 つ）
しかし、<100> は数字ですので、文字に変換しなければなりません。
実数を文字列に変換する rtos 関数を使用します。

```
(setq dist_x (rtos dist_x))
(setq dist_y (rtos dist_y))
```

これで、終点 pt2 の座標が決まりましたので、以下のように連結します。

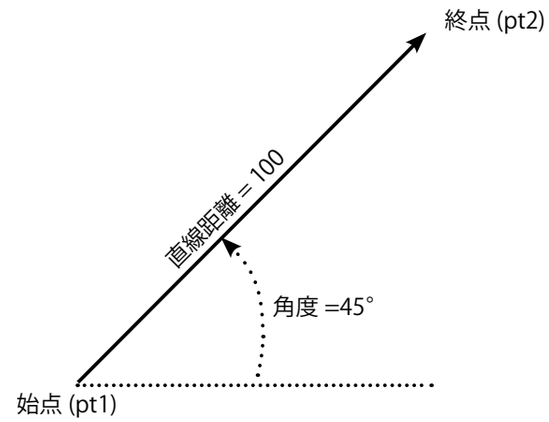
```
(setq  pt2 (strcat  "@"  dist_x  ","  dist_y))      → strcat 関数は、文字を連結する関数です。
```

マウスで始点 (pt1) を指示し、終点 (pt2) は計算で決まりましたので、線分コマンドを使用して次のように記述します。

```
(command  "LINE"  pt1  pt2  "")
```

この例のように、ユーザーがキーボードから入力した数値を他の文字と連結する時は、数値を文字列に変更しておく必要があります。

angtos 関数	角度（ラジアン）を文字列に変換
極座標入力で線分コマンド (LINE) を使う	



線分コマンドの時、始点をマウスで指示をして、終点の位置をキーボードから極座標入力する場合は、(@ 直線距離 < 角度) の形になります。

LISP のプログラムでは、次のようにユーザーに直線距離と角度を要求します。

```
(setq  dist_xy (getreal  "¥n 直線距離："))          → ユーザーが <100> と入力。
(setq  ang_1   (getangle  "¥n 角度："))             → ユーザーが <45> と入力。
```

キーボードからの入力では、(@100<45) ですから、この形に文字を連結します。（要素は 4 つ）
ここで、直線距離の <100> と角度の <45> を文字に変換しなければなりません。
直線距離は、実数を文字列に変換する rtos 関数を使用します。

```
(setq dist_xy (rtos dist_xy))
```

また、角度（ラジアン）を文字列に変換する angtos 関数を使用します。

```
(setq ang_1 (angtos  ang_1))
```

これで、終点 pt2 の座標が決まりましたので、以下のように連結します。

```
(setq  pt2 (strcat  "@"  dist_xy  "<"  ang_1))      → strcat 関数は、文字を連結する関数です。
```

マウスで始点 (pt1) を指示し、終点 (pt2) は計算で決まりましたので、線分コマンドを使用して次のように記述します。

```
(command  "LINE"  pt1  pt2  "")
```

この例のように、ユーザーがキーボードから入力した数値を他の文字と連結する時は、数値を文字列に変更しておく必要があります。

AutoLISP の関数

4 座標系同士の変換関数

TRANS	ある座標系から別の座標系に、点 (または変位) を変換します。

書式　(trans pt from to [disp])

pt は 3D 点または 3D 変位（ベクトル）を表す 3 つの実数のリスト

from は座標系コード 1

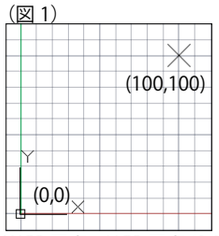
to は座標系コード 2

(Trans 座標 座標系コード 1 座標系コード 2 [切り替え])	
引　数	説　明
座標	基準点（変移）
座標系コード 1	基準点を表す座標系コード
座標系コード 2	変換先の座標系コード
切り替え	最後のオプション引数では、<T> か < 非 nil> のとき、最初の引数は変移と見なされます。省略されるか <nil> のときは点と見なされます。

次の表に、to 引数と from 引数に使用できる有効な整数コードを示します。

座標系コード	
コード	座 標 系
0	ワールド (WCS)
1	ユーザー (現在の UCS)
2	ディスプレイ。コード 0(ゼロ) または 1 と組み合わせて使用する場合、現在のビューポートの DCS、コード 3 と組み合わせて使用する場合、現在のモデル空間ビューポートの DCS
3	ペーパー空間 DCS、PSDCS(コード 2 専用)

(図 1)



WCS 原点 =UCS 原点 (0,0)

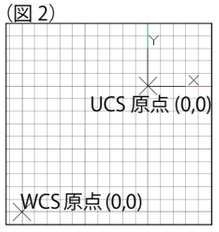
(図 1) は、WCS と UCS が一致した座標において、(100,100) の位置に点 (point) があります。

この点情報を取得すると以下のように表示されます。

((-1 . < 図形名 : 7ef03550>) (0 . "POINT") (330 . < 図形名 : 7ef01cf8>) ・・・(中略)
(10 100.0 100.0 0.0) (210 0.0 0.0 1.0) (50 . 0.0))

(10 100.0 100.0 0.0) から点の座標は (100,100) と確認できます。

(図 2)



今、(図 2) のように UCS の原点を点の座標 (100,100) に設定したとき、作図の原点 (0,0) は点の位置に移動します。

再度、この点情報を取得すると以下のように表示されます。

((-1 . < 図形名 : 7ef03550>) (0 . "POINT") (330 . < 図形名 : 7ef01cf8>) ・・・(中略)
(10 100.0 100.0 0.0) (210 0.0 0.0 1.0) (50 . 0.0))

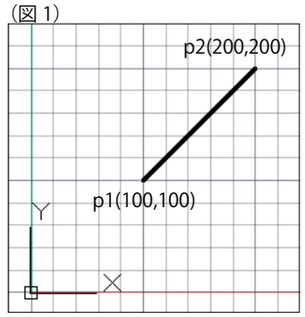
しかし、点の座標は以前と同じ (100,100) です。

これにより、ユーザーがマウスやキーボードから入力する座標値は UCS 座標であり、作成された図形が保持する座標値は WCS 座標であることがわかります。

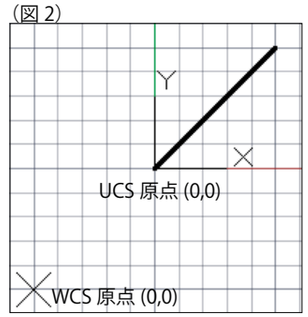
UCS を切り替えて作図する

① (図 1) は WCS と UCS が一致した座標系において、②今、UCS の原点を p1 の位置に設定し、p1(100,100) から p2(200,200) に作図した線分です。線分 (p1-p2) の中点から円を作図します。

(図 1)



(図 2)

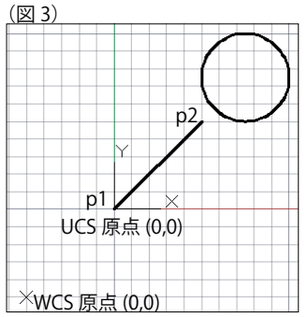


③ (図 1) より、円の中心は (150,150)、半径を <50> として、LISP で以下のように記述します。

(command "CIRCLE" "150,150" 50)

線分の中点に作図するつもりでしたが、右上に大きく離れてしまいました。

(図 3)



作図の途中で UCS 座標系を変更したときは、trans 関数で対象となる図形（図 3 では線分）の座標を変換する必要があります。

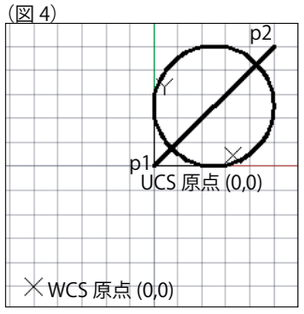
この場合、円を作図する前に点 p1 と点 p2 の座標を次のように変換します。

(setq p1 (trans p1 0 1)) ;WCS から UCS へ変換
(setq p2 (trans p2 0 1)) ;WCS から UCS へ変換

④線分の端点 (p1 と p2) を trans 関数で WCS から UCS に変換したあと、円を作図すると期待通りの円が作図できます。(図 4)

このように、UCS を変更する前の図形に関係して作図するときは、その図形を現在の UCS 座標系に変換する必要があります。

(図 4)



⑤一連の作業が終わった時は、元の座標系 (WCS) に戻すことが必要です。(command "UCS" "W")

Point!

ユーザーが AutoCAD を操作しているときは、現在の UCS 座標で行っていますが、作成したデータは WCS 座標や OCS 座標（平面図形の場合）で保存されます。

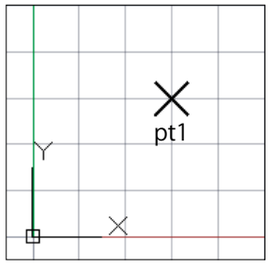
lisp 関数のユーザー対話型の関数 (getpoint、entsel 等）の返す座標値は現在の UCS 座標です。現在の座標系が WCS の場合は、現在の UCS 座標が WCS 座標と一致していることになります。

trans 関数①	ユーザー座標系の原点から作図をする

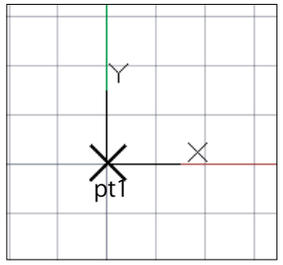
原点からの座標が判らない位置 (×印) から作図を開始する場合は、以下のように行います。

①作図を開始したい位置 pt1(×印) にユーザー座標系の原点を移動します。
② UCS アイコンが移動したことが、画面上で確認できます。
③×印が原点 (0,0) に変更になりましたから、この座標を原点として作図を開始します。
終了したときは、再度 UCS コマンドでワールド座標系 (W) に戻します。

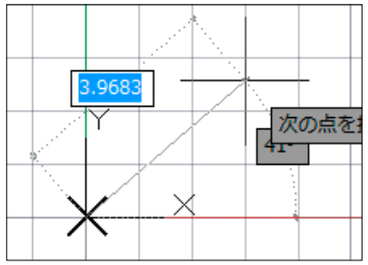
Step1



Step2



Step3



この動作を AutoLISP での記述は以下のようになります。

Step1 ー UCS コマンドで、×印の位置 (pt1) に原点を移動する。❶
(setq pt1 (getpoint "¥n 新しい原点を指示：")) → getpoint 関数で、先に移動する位置を取得。

Step2 ー trans 関数で、ワールド座標系からユーザー座標系に変更する。❷
(command "UCS" pt1 "") → UCS コマンドで、pt1 を新しい原点にします。

Step3 ー×印の新しい原点（ユーザー座標系）から作図を開始する。❸
(command "LINE" pt1 pause "") → 線分コマンドで作図します。

Step4 ー終了するときは、ワールド座標系に戻す。❹
(command "UCS" "W") → 作図が終了すると、ワールド座標系に戻します。

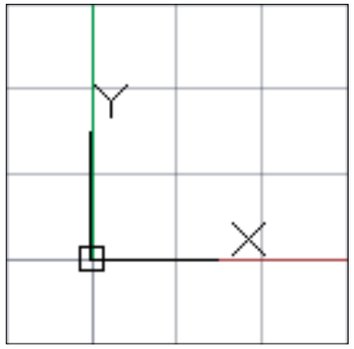
ユーザー座標系に変更した場合は、trans 関数で座標系の変換が必要です。


```
(defun C:ucs_o ( / pt1)
❶ (setq pt1 (getpoint "¥n 新しい原点を指示：")) → Step1
❷ (command "UCS" pt1 "") ;UCS の原点を pt1 に移動 → Step2
❸ (command "LINE" pt1 pause "") ; 線分を作図 → Step3
❹ (command "UCS" "W") ; ワールド座標系に戻す → Step4
) ;end
```

番号	関数名	説　明
①	getpoint	マウスで指示した点を、変数 <pt1> に代入します。
②	"UCS" pt1 ""	ユーザー座標系の原点を pt1 に移動します。終了の <""> は必要です。
③	"LINE" pt1 pause ""	pause はユーザーの指示待ちです。<""> で線分コマンドは終了します。
④	"UCS" "W"	最後に元のワールド座標系に戻します。

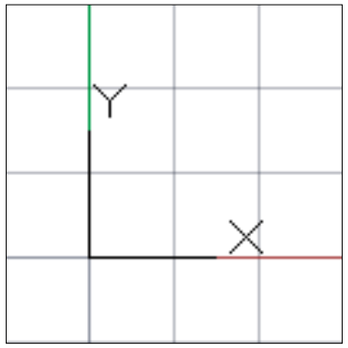
☞getpoint は P1-150 を参考

ワールド座標系の < 原点 >

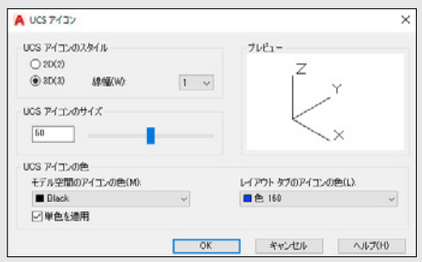


UCS アイコンが変わります！

平行移動したユーザー座標の原点

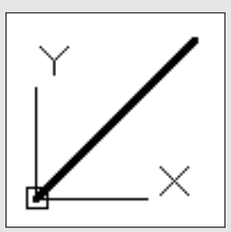


Point!

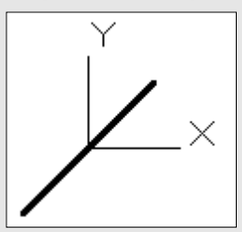


[表示] → [表示設定] → [UCS アイコン] → [プロパティ] から UCS アイコンのスタイルの確認と変更ができます。

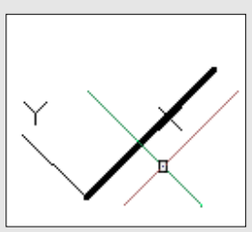
[UCS は原点の位置]



[UCS を平行移動]



[UCS をオブジェクトに合わせる]

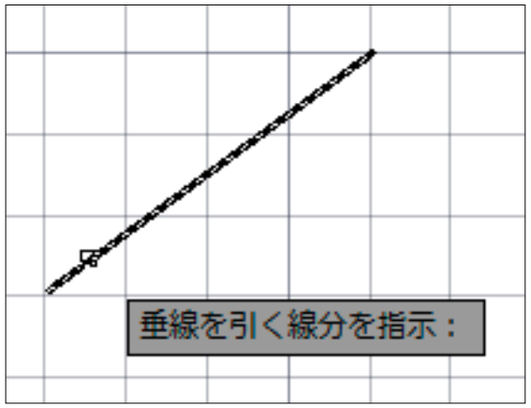


trans 関数②	線分の中点から垂直線を作図する

① UCS コマンドで線分を X 軸に変更する。
②垂直線の始点を取得するために、斜めの線分の中点を指示する。
③直交モードに変更する。
④マウスが直交モードに固定されるので、そのまま線分の終点を指示する。
⑤ UCS コマンドでワールド座標系に戻し、直交モードも解除する。

この動作を AutoLISP では以下のように記述します。

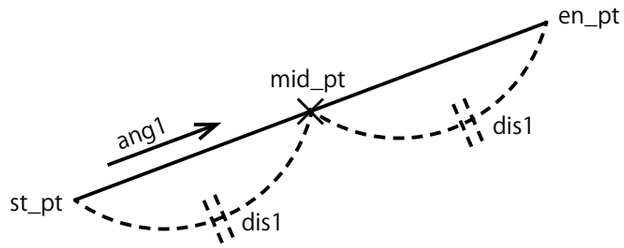
Step1 ー① (entsel "¥n 垂線を引く線分を指示：") ❶
" 垂線を引く線分を指示:" のメッセージを表示して、垂線を引きたい線分を指示します。
変数 <en1> には、図形名と指示した位置の座標が代入されます。
(< 図形名 : 7ef03990> (1369.08 1282.59 0.0))
② en1 は図形名と座標値のリストですから、car 関数で前半の図形名を取得します。❷
(car en1) → < 図形名 : 7ef03990>
③この図形名の中身を展開する関数は entget です。❷
((-1 . < 図形名 : 7ef03990>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A2") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 1289.02 1245.67 0.0) (11 2028.05 1642.56 0.0) (210 0.0 0.0 1.0))



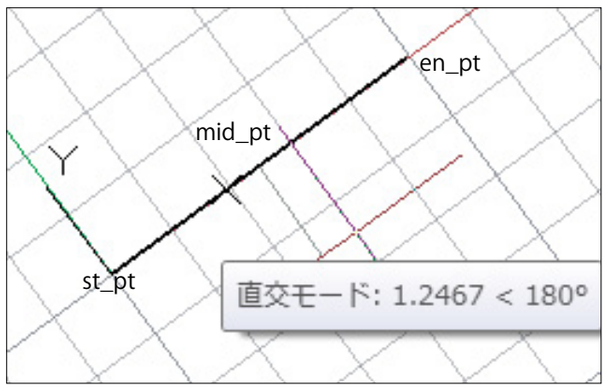
Step2 ー①カーソルを直交モードにして水平・垂直の線分を作成するために、ユーザー座標系を変更して線分を X 軸にします。❸
("UCS" "OB" en1) → "OB" は選択した線分を X 軸にします。
マウスで選択した位置に近い方の端点が原点になります。遠い方の端点が、X 軸の正方向になります。

②線分の始点の座標を取得します。❹
始点の座標は、図形名の中では (10 1289.02 1245.67 0.0) です。
(10 X 座標 Y 座標 Z 座標) → <10> は始点座標であることを表しています。
cdr 関数で座標値を取り出します。 → (cdr (assoc 10 en2))
③取り出した座標値をユーザー座標系に変更します。❺
(trans 現在の座標値 WCS<0> UCS<1>) → (trans st_pt 0 1)
④同様にして、線分の終点の座標も取得します。❻❼

Step3 ー①線分の始点から終点への角度を計算し、変数 <ang1> に代入します。❽
角度を計算する関数は angle です。 → (angle st_pt en_pt)
②線分の始点から終点までの距離（長さ）の半分を計算し、変数 <dis1> に代入します。❾
距離を計算する関数は distance です。 → (/ (distance st_pt en_pt) 2)
③始点 (st_pt) から、角度 ang1、距離 dis1 の位置が、線分の中点 (mid_pt) になります。❿
ただし、この場合の始点 (st_pt) は原点 (0.0 , 0.0) と同じになります。



Step4 ー①カーソルを直交モードに固定します。⓫
(setvar "ORTHOMODE" 1) → <1> は直交モード ON、<0> は OFF です。
②線分コマンドで中点を指示し、終点は pause 関数でユーザーの指示待ちです。⓬



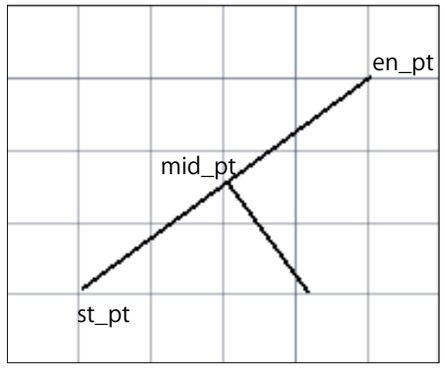
直交モードは、P1-170 を参考

Step5 ー①ユーザー座標系をワールド座標系に戻します。⓭

(command "UCS" "W")

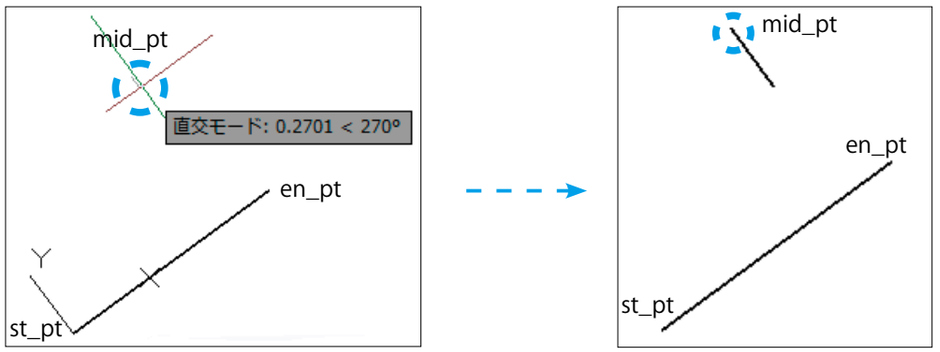
②直交モードも解除します。⓮

(setvar "ORTHOMODE" 0)



(Step2 の③について)

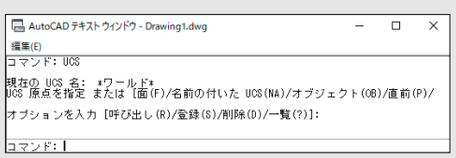
trans 関数でユーザー座標系に変更しない場合は、下図のように予想外の結果になります。



Point!

UCS を変更した後は元の UCS に戻す必要がありますが、AutoCAD のコマンド ラインに表示される UCS コマンドのオプションから UCS を切り替えます。

[面 (F)/ 名前の付いた UCS(NA)/ オブジェクト (OB)/ 直前 (P)/ ビュー (V)/ ワールド (W)/X/Y/Z/Z 軸 (ZA)]



AutoLISP の中で、多くの座標系を使用する場合は、名前を付けておきます。

[UCS] → [NA] から [登録 (S)] をします。

この UCS を使用するときは、[UCS] → [NA] から [呼び出し (R)] を選択します。


```
(defun C:sui_lin ( / en1 en2 st_pt en_pt ang1 dis1 mid_pt)
❶ (setq en1 (entsel "¥n 垂線を引く線分を指示：")) ; 線分を選択
❷ (setq en2 (entget (car en1))) ; 線分の情報を取得
  ; 選択した線分を X 軸に UCS 変換する
❸ (command "UCS" "OB" en1) ; 線分をユーザー座標系の X 軸に変更
❹ (setq st_pt (cdr (assoc 10 en2))) ; 始点の座標を取得
❺ (setq st_pt (trans st_pt 0 1)) ; ユーザー座標系に変更
❻ (setq en_pt (cdr (assoc 11 en2))) ; 終点の座標を取得
❼ (setq en_pt (trans en_pt 0 1)) ; ユーザー座標系に変更
  ; 中点座標を計算する処理
❽ (setq ang1 (angle st_pt en_pt)) ; 始点から終点への角度を計算（0° か 180°）
❾ (setq dis1 (/ (distance st_pt en_pt) 2)) ; 始点と終点の距離の半分を計算
❿ (setq mid_pt (polar (list 0.0 0.0) ang1 dis1)) ; 中点の位置を計算
  ; 直交モードを ON にして線分を作図
⓫ (setvar "ORTHOMODE" 1) ; 直交モードをオン
⓬ (command "LINE" mid_pt pause "") ; 中点から線分を作図
  ; 座標系と直交モードを元に戻す
⓭ (command "UCS" "W") ; 元のワールド座標系に戻す
⓮ (setvar "ORTHOMODE" 0) ; 直交モードをオフ
  (princ)
 );end
```

→ Step1 (❶❷)
→ Step2 (❸❹❺❻❼)
→ Step3 (❽❾❿)
→ Step4 (⓫⓬)
→ Step5 (⓭⓮)

番号	関数名	説　明
①	entsel	entsel 関数で X 軸にする線分を選択します。
②	entget (car en1)	en1 は、図形名と座標点の組み合わせなので、図形名だけを取得。
③	"UCS" "OB" en1	UCS コマンドで、選択した線分を X 軸に変更します。
④	cdr (assoc 10 en2)	en2 の図形情報から線分の始点の座標を取得します。
⑤	trans st_pt 0 1	始点の座標をユーザー座標系に変更します。
⑥	cdr (assoc 11 en2)	en2 の図形情報から線分の終点の座標を取得します。
⑦	trans en_pt 0 1	終点の座標をユーザー座標系に変更します。
⑧	angle st_pt en_pt	始点から終点の角度をラジアンで取得します。(0° か 180°)
⑨	/ (distance st_pt en_pt) 2	始点から終点までの距離の半分（中点）を計算します。
⑩	polar (list 0.0 0.0) ang1 dis1	ユーザー座標系の原点（始点）と終点の中点座標を計算します。
⑪	"ORTHOMODE" 1	直交モードに切り替えます。
⑫	"LINE" mid_pt pause ""	線分コマンドで中点から始めます。終点は pause でユーザー待ち。
⑬	"UCS" "W"	座標系をワールド座標系に戻します。
⑭	"ORTHOMODE" 0	直交モードをオフにします。

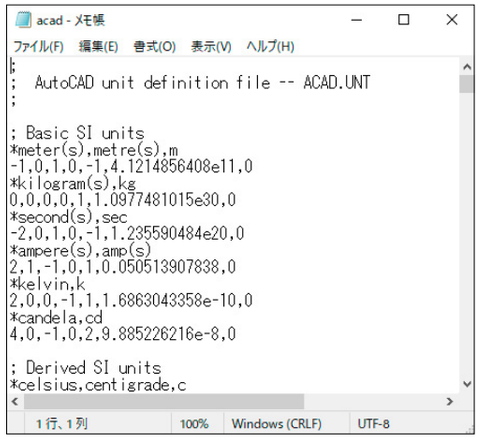
entsel は P1-210、getget は P1-218 を参考

5 計測単位同士の変換関数

acad.unt を使った単位変換

acad.unt ファイルは、マイルからキロメートル、華氏から摂氏など、実際に使用される単位間のさまざまな変換を定義しています。cvunit 関数は、ある単位系で表された値を受け取り、別の単位系に値を変換して返します。2 種類の単位系は、acad.unt で定義されている単位の式を表す文字列によって指定されます。

acad.unt ファイル



cvunit 関数は、互換性のない単位間の変換は行いません。
たとえば、インチからグラムへの変換は行いません。

cvunit 関数を使って単位変換

①距離

インチからメートルへ変換

(cvunit 100 "inches" "meters") → 2.54

メートルからインチへ変換

(cvunit 2.54 "meters" "inches") → 100.0

②角度（円の全体角度 360° を <1> とする）

円周率へ変換

(cvunit 1 "circle" "radian") → 6.28319（2 π）

度（十進法）へ変換

(cvunit 1 "circle" "degree") → 360.0

③時間

1 日を分へ変換

(cvunit 1 "day" "minute") → 1440.0

分を日へ変換

(cvunit 1440 "minute" "day") → 1.0

単位の変換

㎡（平米）を坪面積に変換してみましょう。（1 ㎡= 0.3025 坪）

```
(setq heibei (getreal " 何㎡ですか？ "))          → キーボードから <100> を入力します。
(setq tubo  ( * heibei  0.3025))                  → ㎡に 0.3025 を掛けて、坪数を計算します。
(setq s_heibei (rtos heibei 2 1))                 → ㎡の実数値を文字列に変換します。
(setq s_tubo  (rtos tubo  2 1))                   → 坪数の実数値を文字列に変換します。
(setq ans (strcat s_heibei " ㎡は " s_tubo " 坪です。"))   → 文字列を連結します。(strcat 関数 )
(prompt ans)                                      → コマンド ラインに結果を表示させます。
100 ㎡は 30.3 坪です。                            → コマンドラインに <100 ㎡は 30.3 坪です。> が表示されます。
```

坪面積を㎡（平米）に変換してみましょう。（1 坪= 3.30578 ㎡）

```
(setq tubo (getreal " 何坪ですか？ "))            → キーボードから <100> を入力します。
(setq heibei ( / tubo  0.3025))                   → 坪を 0.3025 で割って、㎡を計算します。
(setq s_tubo (rtos tubo 2 1))                     → 坪数の実数値を文字列に変換します。
(setq s_heibei  (rtos heibei 2 1))                → ㎡の実数値を文字列に変換します。
(setq ans (strcat s_tubo " 坪は " s_heibei " ㎡です。"))   → 文字列を連結します。(strcat 関数 )
(prompt ans)                                      → コマンド ラインに結果を表示させます。
100 坪は 330.6 ㎡です。                           → コマンドラインに <100 坪は 330.6 ㎡です。> が表示されます。
```

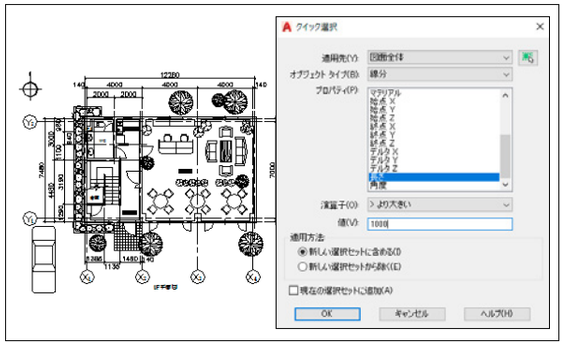
下表は距離と面積の換算表です。

距　離					
メートル m	キロメートル km	インチ in	フィート ft	ヤード yd	マイル mile
1	0.001	39.3701	3.28084	1.09361	0.000621
1,000	1	39,370.10	3,280.84	1,093.61	0.621371
0.0254	0.000025	1	0.083333	0.027777	0.000015
0.3048	0.0003	12	1	0.33333	0.000189
0.9144	0.000914	36	3	1	0.000568
1,609.34	1.609344	63,360	5,280	1,760	1

面　積					
平方メートル m2	アール a	平方キロ km2	平方フィート ft2	平方ヤード yd2	エーカー ac
1	0.01	0.000001	10.7639	1.19599	0.000247
100	1	0.0001	1,076.39	119.599	0.02471
1,000,000	10,000	1	10,764,263	1,195.99	247.11
0.0929	0.000928	-	1	0.111111	0.00002
0.836127	0.008361	0.0000008	9	1	0.000206
4,046.86	40.4686	0.004047	43,560	4,480	1

第 6 節　比較関数

下図は AutoCAD の [クイック選択] のダイアログです。
このダイアログでは、図面の中の線分を長さを指定して選択しようとしています。



対象となる図形ごとに使用する比較演算関数は異なります。
例えば、円を比較するのに終点の座標や角度の比較はありえません。
比較演算を行うときは、このように対象とする図形の種類や目的に合った演算関数を選ぶ必要があります。

この節では、修飾子とか論理演算子といった難しい言葉が出てきますが、使用方法は通常の数式と変わるところはありません。比較演算関数は次節の条件関数と連動して良く使用されますし、第 3 章と第 4 章の中心になる関数のグループです。

使用頻度の高い AutoLISP の 比較演算関数を、次の表に示します。

①比較演算関数（修飾子）	
関　数	説　明
(= (equal to) numstr [numstr] ...)	すべての引数が数値的に等しい場合は T を返し、それ以外の場合は nil を返します。
(/= (not equal to) numstr [numstr] ...)	引数が数値的に等しくない場合は T を返し、引数が数値的に等しい場合は nil を返します。
(< (less than) numstr [numstr] ...)	各引数がその右側の引数よりも数値的に小さい場合は T を返し、それ以外の場合は nil を返します。
(<= (less than or equal to) numstr [numstr] ...)	各引数がその右側の引数よりも数値的に小さいか等しい場合は T を返し、それ以外の場合は nil を返します。
(> (greater than) numstr [numstr] ...)	各引数がその右側の引数よりも数値的に大きい場合は T を返し、それ以外の場合は nil を返します。
(>= (greater than or equal to) numstr [numstr] ...)	各引数がその右側の引数よりも数値的に大きいか等しい場合は T を返し、それ以外の場合は nil を返します。

②比較演算関数（論理演算子）	
関　数	説　明
(and [expr ...])	指定された引数の AND(論理積) を返します。
(not item)	指定された項目の評価が nil になるかどうかを調べます。
(or [expr...])	式のリストの OR(論理和) を返します。
(null item)	指定された項目の内容が nil かどうかを調べます。

③比較演算関数（式）	
関　数	説　明
(eq expr1 expr2)	2 つの式が同一物かどうかを調べます。
(equal expr1 expr2 [fuzz])	2 つの式の評価結果が等しいかどうかを調べます。

1　比較演算関数（修飾子）

=	全ての引数が数値的に等しい場合に、T を返します。（等しくなければ、nil）

```
(setq a 2)
(= a 2)          → T
(= a 3)          → nil
```

/=	各引数が等しくなければ、T を返します。（等しければ、nil）

```
(setq a 2)
(/= a 2)         → nil
(/= a 3)         → T
```

<	第 1 引数が第 2 引数より小さければ、T を返します。（大きければ、nil）

```
(setq a 2)
(< a 3)          → T
(< a 1)          → nil
```

<=	第 1 引数が第 2 引数より小さいか等しい場合に、T を返します。

```
(setq a 2)
(<= a 2)         → T
(<= a 1)         → nil
```

>	第 1 引数が第 2 引数より大きければ、T を返します。（小さければ、nil）

```
(setq a 2)
(> a 1)          → T
(> a 2)          → nil
```

>=	第 1 引数が第 2 引数より大きいか等しい場合に、T を返します。

```
(setq a 2)
(>= a 2)         → T
(>= a 3)         → nil
```

2 比較演算関数（論理演算子）

AND	各引数がすべてＴの場合に、Ｔを返します。

```
(setq a 2 b 3)
(and (= a 2) (= b 3))          →T
(and (= a 2) (= b 4))          →nil
```

OR	式を順番に評価して、nil でない式が 1 つでもあれば、Ｔを返します。

```
(setq a 2 b 3)
(or (= a 2) (= b 4))           →T （どちらかが正しいので、T）
(or (= a 3) (= b 4))           →nil（どちらも誤りなので、nil）
```

NOT	シンボルは nil の場合に、Ｔを返します。

```
(setq a 2 b nil)
(not a)                        →nil
(not b)                        →T
```

NULL	指定された引数が、nil であるかどうかを判断します。

```
(setq a 2  b "AutoLISP"  c nil)
(null a)                       →nil（空でないから nil）
(null b)                       →nil（空でないから nil）
(null c)                       →T  （空だから T）
```

データ内容を判定する関数		
関数	判定方法	変換結果
atom	データがリストならば	nil
boundp	アトムが値を持っていれば	T
listp	データがリストならば	T
minusp	データが負の数値ならば	T
not	データが値を持っていなければ	T
null	リストが値を持っていなければ	T
numberp	データが数値ならば	T
zerop	データがゼロの数値ならば	T

P で終わっている修飾子 (boundp、listp、minusp、zerop) の P は、修飾子 (Predicate) の頭文字です。

修飾子と論理演算子

修飾子と論理演算子はテスト式で使用します。
修飾子と論理演算子も関数で、評価結果として真 (True)、または偽 (nil) を返します。

① (> 5 8)
修飾子 > は、左の数値が右の数値より大きいかどうかをテストします。この場合、5 は 8 より大きくありませんから、評価値は nil になります。

② (> 5 8 7 2)
修飾子 (>、<、>=、<=) は、2 つ以上の引数を持つことができます。
引数が 2 つ以上ある場合は、2 番目以降のすべての引数が 1 番目の引数より大きい時に <T> を返します。上記の場合は、5 > 8 > 7 > 2 ではありませんから、<nil> を返します。

③論理演算子 (and、not、null、or) は、値として <T> か <nil> を返す点では修飾子と同じです。
しかし、論理演算子では修飾子が評価した値に対して、さらに評価を加えることが可能です。
（例） (not (> 5 8))
T
<not> は引数が <nil> の場合に <T> を返しますから、結果は <T> になります。

3 比較演算関数（式）

pt1、pt2、pt3、pt4 に以下のように値をセットします。

```
(setq pt1 '(10 20 30))         →(10 20 30)
(setq pt2 '(10 20 30))         →(10 20 30)
(setq pt3 pt1)                 →(10 20 30)
(setq pt4 pt2)                 →(10 20 30)
```

eq	同一のオブジェクト (つまりアドレスが等しい) 場合に真を返します。

```
(eq pt1 pt2)                   →nil（参照先 < アドレス > が違う）
(eq pt1 pt3)                   →T （参照先 < アドレス > が同じ）
(eq pt2 pt4)                   →T （参照先 < アドレス > が同じ）
(eq pt3 pt4)                   →nil（参照先 < アドレス > が違う）
```

equal	2 つの式の評価結果が等しい場合に真を返します。

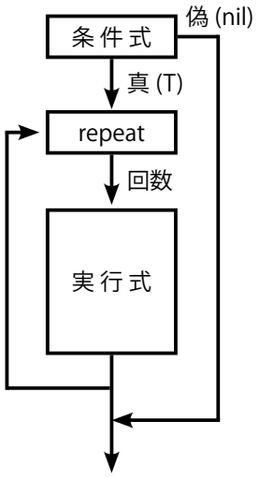
```
(equal pt1 pt2)                →T （評価結果 < 値 > が同じ）
(equal pt1 pt3)                →T （評価結果 < 値 > が同じ）
(equal pt2 pt4)                →T （評価結果 < 値 > が同じ）
(equal pt3 pt4)                →T （評価結果 < 値 > が同じ）
```

第 7 節　条件関数（繰り返し関数・分岐関数）

繰り返し関数 (repeat、while)

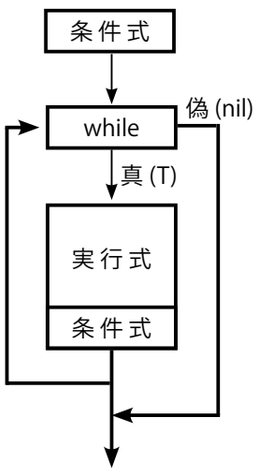
①同じ処理を繰り返して行うとき、繰り返しの回数が分かっている場合に repeat 関数を使います。
repeat の前に回数（正の整数）をセットし、その後に繰り返しの処理を記述します。

```
;円を作図する回数を入力します。
(setq kaisuu (getint "¥n 円の回数 "))
;回数分だけ繰り返します。
(repeat kaisuu
;円の中心を指示します。
  (setq center (getpoint "¥n 円の中心 "))
;円の半径を入力します。
  (setq hankei (getreal "¥n 円の半径 "))
;円を作図します。
  (command "CIRCLE" center hankei)
;kaisuu 回繰り返します。
);repeat
```



②同じ処理を繰り返して行うとき、繰り返しの回数が分かっていない場合に while 関数を使います。
while の前に条件式をセットし、真 (T) であれば実行式を処理します。
そして、実行式の末尾に再度条件式をセットし、最初の while に戻ります。

```
;円の半径を入力します。（円の大きさは固定）
  (setq hankei (getreal "¥n 円の半径 "))
;円の中心を指示します。
  (setq center (getpoint "¥n 円の中心 "))
;center が入力されていれば実行します。
(while (/= center nil)
;円を作図します。
  (command "CIRCLE" center hankei)
;円の中心を指示します。
  (setq center (getpoint "¥n 円の中心 "))
;center が入力されなければ終了します。
);while
```



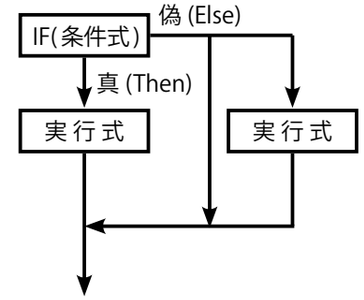
AutoLISP の 繰り返し関数を、次の表に示します。（順不同）

条件関数（条件に合う間、コマンドを繰り返す関数）	
関　数	説　明
(repeat int [expr ...])	指定された回数だけ各式を評価し、最後の式の値を返します。
(while testexpr [expr ...])	テスト式を評価して nil でなければ、他の式を評価します。 テスト式の評価が nil になるまでこの処理を繰り返します。

分岐関数 (if、cond)

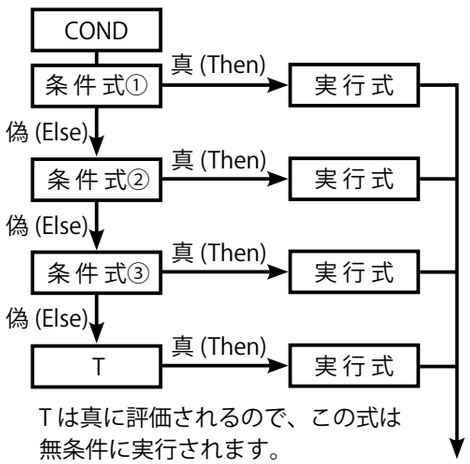
① if は条件式を評価し、その結果が真ならば then 式を実行します。条件式が偽であれば else 式を実行します。else 式は省略することができます。その場合、条件式で偽と判定されると nil を返します。これを図に示すと、以下のようになります。

```
;線分を作図するかどうかを尋ねます。（<Y> か <N> を入力）
(setq ans (getstring "¥n 線分を作図しますか？ <Y>/<N>)
;もし、<Y> であれば、以下を実行します。
(if (= ans "Y")
  (progn  ;始点と終点の座標を取得します。
    (setq pt1 (getpoint "¥n 始点を指示 "))
    (setq pt2 (getpoint "¥n 終点を指示 "))
    (command "LINE" pt1 pt2 "")  ;線分を作図します。
  );progn
);if
```



② cond は複数の式を引数として受け取ります。各式の先頭には条件をチェックする記述があり、条件が成立した場合、その実行式を評価します。条件が不成立であれば、次の式に移ります。たとえば、条件式① が不成立であれば、次の式に移り条件式② をチェックします。条件が成立したときは、同じ式にある実行式を評価します。そして、いちばん最後に評価された実行式の返り値が cond の返り値となります。
したがって、一度条件式が選択されたら、それ以降の式は評価されません。もし、どの条件式も不成立であれば、cond は nil を返します。これを図に表すと以下のようになります。

```
;どの図形を作図するかを尋ねます。（<L> か <C> か <R> を入力）
(setq ans (getstring "¥nLine  Circle  Rectang"))
(cond
;ans が "L" であれば、線分を作図します。
  ((= ans "L") (command "Line" pause pause ""))
;ans が "C" であれば、円を作図します。
  ((= ans "C") (command "Circle" pause pause))
;ans が "R" であれば、長方形を作図します。
  ((= ans "R") (command "Rectang" pause pause))
;a 上記以外の場合、[ 終了します。] と表示します。
  (    T      (prompt "¥n 終了します。"))
);cond
```



T は真に評価されるので、この式は無条件に実行されます。

AutoLISP の 分岐関数を、次の表に示します。（順不同）

条件関数（条件によって処理を分岐する関数）	
関　数	説　明
(if testexpr thenexpr [elseexpr])	条件に応じて式を評価します。
(cond [(test result ...) ...])	条件を順番に評価して、T の式を実行します。

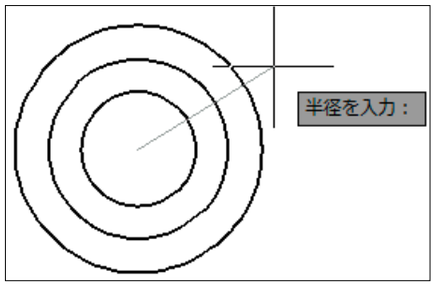
1 条件関数（条件に合う間、コマンドを繰り返す関数）

REPEAT	指示した回数だけ、繰り返して実行します。

```
(setq  count  1)                    → count の初期値をセットします。
(repeat  5                          → 5 回繰り返します。
    (setq   a  (1+  count ))        → count に 1 加算します。
    (setq  count (1+ count))        → count の数を 1 づつ増やします。
)                                   → 6 （変数 <a> と <count> の値）
```

repeat 関数	指定した回数だけ同心円を作図する

①同心円の中心を指示します。
　→ (getpoint "¥n 円の中心を指示：")
②円の個数を入力します。
　→ (getint "¥n 円の個数を入力：")
③円の個数分、繰り返します。
　→ (repeat ct1)
④半径を入力します。
　→ (getdist cp1 "¥n 半径を入力：")
⑤円を作図します。
　→ 中心は <cp1>、半径は <rad>




```
(defun C:ReCircle( / cp1  ct1 rad)
❶   (setq cp1  (getpoint  "¥n 円の中心を指示："))  ; 円の中心座標を指示
❷   (setq ct1  (getint  "¥n 円の個数を入力："))  ; 作図する円の数を入力
❸     (repeat  ct1  ; 指定した数だけ作図を繰り返す
❹          (setq  rad (getdist  cp1  "¥n 半径を入力："))  ; 半径を取得
❺          (command "CIRCLE"  cp1  rad)  ; 円を作図する
    );repeat
);end
```

番号	関数名	説　明
①	getpoint	円の中心座標を <cp1> にセットします。
②	getint	円の個数を <ct1> にセットします。
③	repeat ct1	<ct1> の回数だけ、円を作図します。
④	getdist cp1	円の半径を <rad> にセット（cp1 からラバーバンドが表示されます）
⑤	("CIRCLE" cp1 rad)	円の中心 <cp1> と半径 <rad> で円を作図します。

Re_Copy.lsp	指定した回数だけ右側へ連続複写する
目的：	円を選択し、複写の回数を入力すると、その回数分だけ円を右側へ直径間隔で複写します。
主要関数：	<entsel><car><entget><getint><polar>

Step1 ー 複写する円を選択します。❶
　(setq ent1 (entget (car (entsel "¥n 円を指示してください。"))))

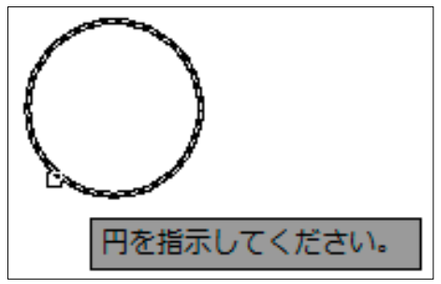
Step2 ー 選択したオブジェクトが円かどうかを判断します。❷❸❹
　(if (/= ent2 "CIRCLE")

Step3 ーオブジェクトが円であれば、複写の回数の入力を聞いてきます。❺
　(setq kai(getint "¥n 何回複写しますか："))

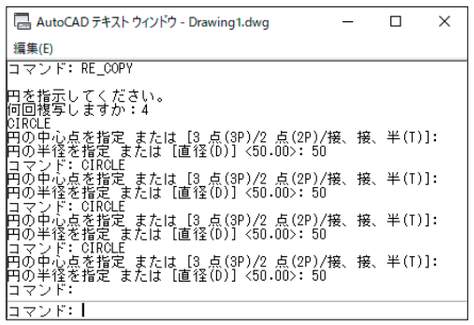
Step4 ー 円の直径（半径の 2 倍）を取得し、右側へ複写する円の中心座標を計算します。❻❼❽
　(setq new_Pt (polar new_Pt 0.0 (* 2 a2)))

Step5 ー 入力した個数分の円を作図します。❾❿⓫
　(command "CIRCLE" new_Pt a2)

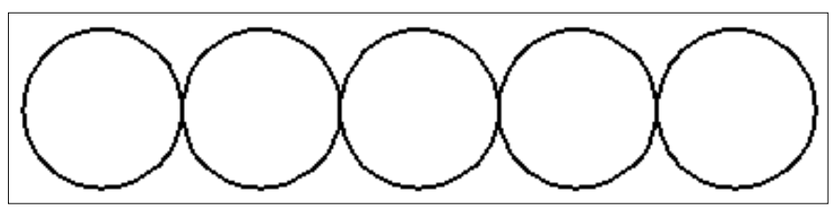
Step1 ー複写する円を選択します。



Step2 ～ 4 ー円を複写する座標を計算処理します。



(Step5) ー指定した回数分だけ円を作図します。（例は、4 回複写）



AutoLISP の関数

```
(defun C:Re_Copy( / ent1 ent2  kai a1 a2)
❶  (setq ent1 (entget (car (entsel "¥n 円を指示してください。" ))))          → Step1
❷  (setq ent2 (cdr (assoc 0 ent1)))   図形のタイプ (CIRCLE) をセット
❸    (if (/= ent2 "CIRCLE" )   円でなければ、メッセージを表示して終了する
❹      (prompt "¥n 選択した図形は円ではありません。" )
         (progn   ;円であれば、以下を実行する
❺        (setq kai(getint "¥n 何回複写しますか : " ))  ;複写の回数をセット
❻        (setq a1 (cdr (assoc 10 ent1)));円の中心座標を取得
❼        (setq a2 (cdr (assoc 40 ent1)));円の半径を取得
❽        (setq new_Pt (polar a1 0.0 (* 2 a2)))  ; 次の円の中心座標を計算
❾          (repeat kai  ;回数分複写する
❿            (command "CIRCLE" new_Pt a2)  ; 円を作図する
⓫            (setq new_Pt (polar new_Pt 0.0 (* 2 a2)))  ; 次の円の中心を計算
                                                          → Step2 ～ 4
                                                          → Step5
           );repeat
         );progn
      );if
   (princ)
) ;end
```

番号	関数名	説　明
①	entsel	entsel で選択した図形の情報を取得します。ent1 には図形の中身がセットされます。
②	(cdr (assoc 0 ent1))	図形 ent1 の図形タイプは グループコード <0> にあります。 cdr 関数で図形のタイプ (線分とか円とか) の情報が得られます。
③	if (/= ent2 "CIRCLE")	ent2 が円かどうかを判定します。
④	prompt	選択した図形が円でなければ、このメッセージを表示して終了します。円であれば、progn 以下に進みます。
⑤	getint	円の複写の回数を取得します。
⑥	(cdr (assoc 10 ent1))	円の中心座標を取得します。
⑦	(cdr (assoc 40 ent1))	円の半径を取得します。
⑧	(polar a1 0.0 (* 2 a2))	次の円の中心座標を計算します。
⑨	repeat kai	kai 数回繰り返します。
⑩	"CIRCLE" new_Pt a2	円を作図します。(中心座標 <new_Pt>、半径 <a2>)
⑪	polar new_Pt 0.0 (* 2 a2)	次の円の中心座標を計算します。

☞ entget は、P1-208、P1-217 を参考
entsel は、P1-208、P1-210 を参考

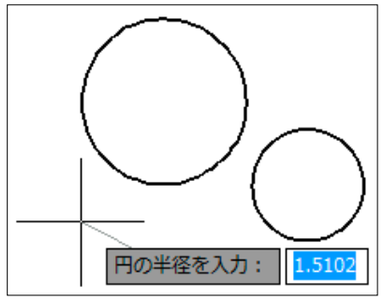
円のグループ コード	
グループコード	意　味
0	オブジェクトの種類 (LINE、CIRCLE、ARC 等)
39	厚さ (省略可能、既定 = 0)
10	中心点 (OCS)　DXF: X 値、APP: 3D 点
20,30	DXF: 始点の Y および Z の値 (OCS)
40	半径
210	押し出し方向 (省略可能、既定 = 0, 0, 1)　DXF: X 値、APP: 3D ベクトル

WHILE	条件が nil になるまで、繰り返して実行します。

```
(setq  count  1)                    → count の初期値をセットします。
(while  (<  count  5)               → count が 5 より小さい時、実行します。
   (setq  a  count)                 → 変数 <a> に <count> の値を代入します。
   (setq  count  (1+  count ))      → count の数を 1 づつ増やします。
)                                   → 4( 変数 <a> の値)、5( 変数 <count> の値)
```

while 関数	指定が中止されるまで円を作図する

①円の中心を指示します。
　→ (getpoint "¥n 円の中心を指示 : ")
②円の半径を入力します。
　→ (getdist cp1 "¥n 円の半径を入力 : ")
　getdist 関数は、マウスからでもキーボードからでも
　入力できます。
③中心と半径が入力された時 (真 <T>)、作図します。
　→ (while dia)
④円を作図します。
　→ 中心は <cp1>、半径は <rad>
⑤円の中心を指示します。
⑥円の半径を入力します。




```
(defun C:WhCircle( / cp1  rad)
❶  (setq cp1  (getpoint  "¥n 円の中心を指示 : "))  ; 円の中心座標を取得
❷  (setq rad  (getdist  cp1  "¥n 円の半径を入力 : "))  ; 円の半径を取得
❸  (while  (and (/= cp1 nil) (/= rad nil))  ; 円の中心と半径が入力されていれば、以下を実行
❹    (command "CIRCLE"  cp1  rad)  ; 円を作図
❺    (setq cp1  (getpoint  "¥n 円の中心を指示 : "))  ; 円の中心座標を取得
❻    (setq rad  (getdist  cp1 "¥n 円の半径を入力 : "))  ; 円の半径を取得
   ) ;while
) ;end
```

番号	関数名	説　明
①	getpoint	円の中心座標を <cp1> にセットします。
②	getdist cp1	円の半径を <rad> にセットします。
③	while	<cp1> と <rad> の両方が入力された時だけ、以下を実行します。
④	"CIRCLE" cp1 rad	円の中心 <cp1> と半径 <rad> で円を作図します。
⑤	getpoint	円の中心の入力を促します。(<cp1> が <nil> の時、このプログラムは終了)
⑥	getdist	円の半径の入力を促します。(<rad> が <nil> の時、このプログラムは終了)

while_off.lsp	間隔の違うオフセットを連続して行う
目的：	線分を作図した後、続けてオフセットコマンドに移ります。 オフセット間隔の入力を続ける限り、オフセットを続けます。
主要関数：	<pause><entlast><getpoint><getdist><OFFSET>

Step1 ー ① "通り芯の一本目を引いてください。" をコマンドラインに表示します。❶

② 1 本目の基準線を作図します。❷

(command "LINE" pause pause "")

pause 関数は、ユーザーが線分の始点と終点を指示するのを待ちます。

③ (setq ent (entlast)) の entlast は最後に作図したオブジェクトを取得します。❸

Step2 ー ①作図した線分をオフセットする側を指示します。❹

(setq pt (getpoint "¥n オフセットする側を指示："))

②オフセット間隔を指定します。❺

(getdist "¥n オフセット間隔：") → getdist 関数は、画面上でも指示できます。

Step3 ー①オフセットの間隔が入力される限り、オフセットを続けます。❻

(while dist・・ → (while (= dist T)・・と同じです。

②オフセットを行います。❼

offset コマンドの書式は、(offset　オフセット距離　オブジェクト　オフセットの側)

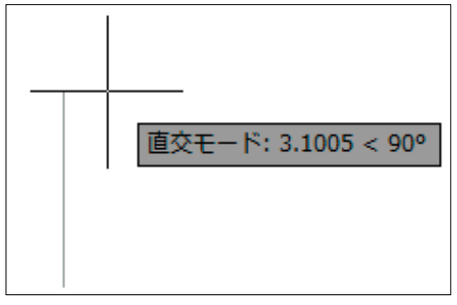
③オフセットの間隔を指示します。❽

(この間隔は、最初の線分 < 基線 > からの距離です。)

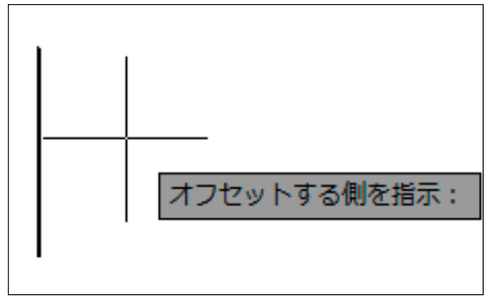
(setq dist (getdist "¥n オフセット間隔："))

※ While 関数の中で、変数 <dist> が <nil（空打ちしない）> でない限り、オフセットを続けます。

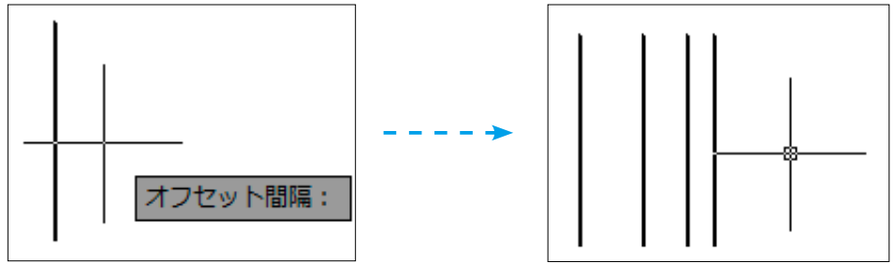
Step1 ー基準線を作図します。



Step2 ーオフセットする方向を指示します。



Step3 ー基準線からのオフセット間隔を入力します。空打ちすると終了します。




```
(defun C:while_off ( / ent pt dist)
; 基準線からの連続オフセット
❶ (prompt "¥n 通り芯の一本目を引いてください。" )  ; メッセージを表示
❷ (command "LINE" pause pause "" )  ; 線分を作図 (pause はユーザー指示待ち )
❸ (setq ent (entlast))  ; 線分を変数 ent に代入（オフセット時に使用）
                                                    → Step1
; あらかじめ、オフセットする側と距離を取得する
❹ (setq pt (getpoint "¥n オフセットする側を指示：" ))  ; 方向を取得
❺ (setq dist (getdist "¥n オフセット間隔：" ))  ; 間隔を取得
                                                    → Step2
;while 関数で、距離が入力される限り、オフセットを続ける
❻ (while dist  ; 距離が入力されていれば、以下を実行
❼   (command "OFFSET" dist ent pt "" )  ; オフセットを行う
❽   (setq dist (getdist "¥n オフセット間隔：" ))  ; 次の距離入力を求める
                                                    → Step3
  ) ;while
(princ)
) ;end
```

番号	関数名	説　明
①	prompt	コマンドラインに表示させる文になります。
②	pause	ユーザーが入力するまで、プログラムをストップします。 pause が 2 回あるのは、線分の始点と終点の入力を待っているからです。
③	entlast	最後に（直前に）作図された図形です。それを変数 ent にセットします。
④	getpoint	ユーザーにオフセットする側を指示させて、その座標を変数 pt にセットします。
⑤	getdist	ユーザーにオフセット間隔を指示させて、その長さを変数 dist にセットします。
⑥	while dist	while (= dist T) と同じです。距離入力があったかどうかチェックしています。
⑦	command	オフセットコマンドを使います。最後の <""> はオフセットの終了になります。
⑧	getdist	ユーザーにオフセット間隔を指示させて、その長さを変数 dist にセットします。 再度間隔の入力を促します。 ⑥に戻り、入力のチェックを行っています。(dist が真である限り、続けます。)

☞ entlast は、P1-209 を参考
getpoint、getdist は、P1-156 を参考

2 条件関数（条件によって処理を分岐する関数）

IF	条件がＴの時、1 番目の式を実行します。（Ｔでない時は、2 番目の式を実行）

```
(setq  a  3)
(if  (= b  a)                    → 変数 <b> の値と <a> を比較します。
     (setq   b  (+  a  1))       →ｂが ａと同じである時の処理。
     (setq   b  (+  a  2))       →ｂが ａと違う時の処理。
)
```

Point!

条件判断に基づいて実行の流れを変える制御関数としての If 関数は、
次のようになります。

条件式が正しければ (真)、Then 式 (最初の式) が実行され、
違っていれば (偽)、Else 式 (次の式) が実行されます。

```
( If  ( 条件式 )
        ( Then の処理 )   ・・・ 条件式が真 (T) のとき、実行する
        ( Else の処理 )   ・・・ 条件式が偽 (nil) のとき、実行する
)
```

ユーザーに数値の 1 か 2 を入力させて、1 だったら線分コマンド、2 だったら円コマンド、
1 でも 2 でもない場合は、長方形コマンドになるプログラムを考えてみましょう。

```
(setq  A  (getint “¥n1 か 2 の数値を入力してください。” ))
(if  ( =  A  1 )                    ;A が <1> かどうかを調べる
     (command  “LINE” )             ; 条件が真 <T> であれば、線分コマンドを実行
);if
(if  ( =  A  2 )                    ;A が <2> かどうかを調べる
     (command  “CIRCLE” )           ; 条件が真 <T> であれば、円コマンドを実行
);if
(if  (and ( /= A  1) ( /= A  2))    ;A が <1> でも <2> でもない時の処理
     (command  “RECTANG” )          ; 条件が真 <T> であれば、長方形コマンドを実行
);if
```


PROGN	IF 式で複数の式を実行します。

```
(setq  a  3)
(if  (= b  a)                    → 変数 <b> の値と <a> を比較します。
  (progn                         →条件式が正しいときの処理
    (setq  c  (+  a  1))         → Then 式 <T>
    (setq  d  (+  a  2))         → Then 式 <T>
  );prog
  (progn                         →条件式が間違っているときの処理
    (setq  c  (-  a  1))         → Else 式 <nil>
    (setq  d  (-  a  2))         → Else 式 <nil>
  );prog
);if
```

複数の式をまとめる <Progn>

実行する処理が複数ある場合は Progn を加えます。
前ページを progn で記述すると以下のようになります。

```
(setq  A  (getint “¥n1 か 2 の数値を入力してください。” ))
(if (or ( =  A  1 ) ( =  A  2 ))    ;A が <1> か <2> のときの処理を記述
  (progn                             ;A が <1> か <2> の場合
    (if  ( =  A  1 )                 ;A が <1> かどうかを調べる
      (command  “LINE” )             ;A が <1> であれば、線分コマンドを実行
      (command  “CIRCLE” )           ;A が <2> であれば、円コマンドを実行
    );if
  );progn
  (progn                             ;A が <1> でも <2> でもない場合
      (command  “RECTANG” )          ; 長方形コマンドを実行
  );progn
);if
```

progn は処理が 1 行でも可能です。

chusin.lsp	選択した円に中心線を作図する
目的：	選択した図形が円の時、中心線を作図します。
主要関数：	<entsel><car><entget><assoc><cdr><polar>

Step1 ー 中心線を作図する円を選択します。

① (setq ent1 (entget (car (entsel "¥n 円を指示してください。")))) ❶

円の図形名を変数 <ent1> にセットします。

((-1 . < 図形名 : 7ef039e0>) (0 . "CIRCLE") (330 . < 図形名 : 7ef01cf8>) (5 . "2AC")
(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle")
(10 800.665 699.811 0.0) (40 . 64.0775) (210 0.0 0.0 1.0))

② (setq ent2 (cdr (assoc 0 ent1))) ❷

図形名 ent1 の図形を取得します。

→ (assoc 0 ent1) から、(0 . "CIRCLE") が取得できます。

→ cdr 関数で、"CIRCLE" が取得できます。

Step2 ー ent2 が円であれば、その中心座標と半径を取得します。❸❹

中心座標 → (setq a1 (cdr (assoc 10 ent1))) ❺

半　　径 → (setq a2 (cdr (assoc 40 ent1))) ❻

Step3 ー円の半径から中心線の長さを計算します。❼❽❾❿

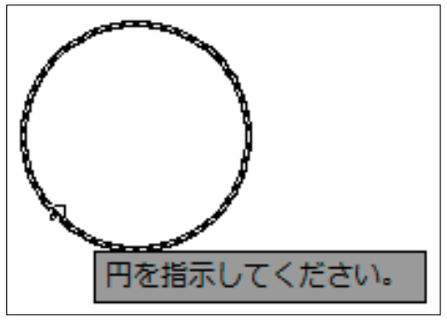
(setq pt1 (polar a1 pi (* a2 1.2))))

→ 中心線は直径の 1.2 倍としています。（円の中心から、180°、0°、90°、270°の方向に半径の 1.2 倍の距離を計算しています。

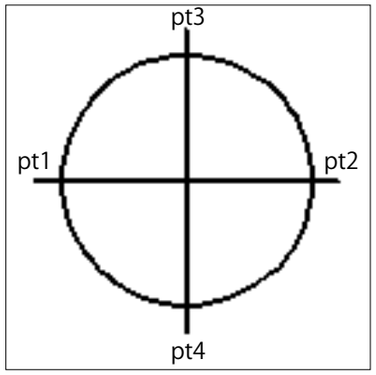
Step4 ー計算に基づいて、中心線を作図します。⓫⓬

(command "line" pt1 pt2 "")
(command "line" pt3 pt4 "")

Step1 ー円を選択します。



Step4 ー中心線を作図します。




```
(defun C:chusin( / ent1 ent2 a1 a2 pt1 pt2 pt3 pt4)
❶  (setq ent1 (entget (car (entsel "¥n 円を指示してください。"))))
❷  (setq ent2 (cdr (assoc 0 ent1))) ; 選択した図形のタイプを取得する
❸  (if (/= ent2 "CIRCLE" ) ; 図形が円でなければ、プログラムを終了する
❹    (prompt "¥n 選択した図形は円ではありません。" )
       (progn  ; 円であれば以下を実行する
❺      (setq a1 (cdr (assoc 10 ent1))) ; 円の中心座標を取得する
❻      (setq a2 (cdr (assoc 40 ent1))) ; 円の半径を取得する
; 円の中心からの長さを計算する
❼      (setq pt1 (polar a1 pi (* a2 1.2))) ;pt1 は a1 から 180° の位置
❽      (setq pt2 (polar a1 0  (* a2 1.2))) ;pt2 は a1 から 0° の位置
❾      (setq pt3 (polar a1 (/ pi 2) (* a2 1.2))) ;pt3 は a1 から 90° の位置
❿      (setq pt4 (polar a1 (/ (* 3 pi) 2) (* a2 1.2))) ;pt4 は a1 から 270° の位置
;2 本の線分を作図する
⓫      (command "line" pt1 pt2 "" ) ; 水平線を作図する
⓬      (command "line" pt3 pt4 "" ) ; 垂直線を作図する
       ) ;progn
  ) ;if
(princ)
) ;end
```

Step1 → Step2 → Step3 → Step4

番号	関数名	説　明
①	ent1	entsel で選択した図形の情報を取得します。ent1 には図形名と座標がセットされます。
②	ent2	図形 ent1 の図形タイプは グループコード <0> にあります。cdr 関数で図形のタイプ (線分とか円とか) の情報が得られます。
③	if (/= ent2 "CIRCLE")	ent2 が円かどうかを判定します。
④	prompt	選択した図形が円でなければ、このメッセージを表示して終了します。円であれば、progn 以下に進みます。
⑤	(cdr (assoc 10 ent1))	円の中心の座標を取得します。
⑥	(cdr (assoc 40 ent1))	円の半径を取得します。
⑦	(polar a1 pi (* a2 1.2))	a1 からの角度は 180°、長さは半径 (a2) の 1.2 倍にセット
⑧	(polar a1 0 (* a2 1.2))	a1 からの角度は 0°、長さは半径 (a2) の 1.2 倍にセット
⑨	(polar a1 (/ pi 2) (* a2 1.2)	a1 からの角度は 90°、長さは半径 (a2) の 1.2 倍にセット
⑩	(polar a1 (/ (* 3 pi) 2) (* a2 1.2))	a1 からの角度は 270°、長さは半径 (a2) の 1.2 倍にセット
⑪	("line" pt1 pt2 "")	pt1 から pt2 へ線分を作図します。
⑫	("line" pt3 pt4 "")	pt3 から pt4 へ線分を作図します。

Point!

このプログラムでは UCS を変更していないので、trans 関数は使っていませんが、既存の図形を使って作図をするときは、UCS 座標系に変換して作図を行う方が安全です。

AutoLISP の関数

COND	条件を順番に評価して、Tの式を実行します。

```
(setq  a  (getint  "¥n<1> か <2> か <3> の数値を入力："))
(cond
    ((= a  1)  (setq   b  (+  a  1)))          → a が <1> の時の処理。
    ((= a  2)  (setq   b  (+  a  2)))          → a が <2> の時の処理。
    ((= a  3)  (setq   b  (+  a  3)))          → a が <3> の時の処理。
    ( T        (setq   b  (+  a  4)))          → a が <1> でも <2> でも <3> でもない時の処理
)
```

Point!

cond 関数の 4 行目に <T> がありますが、<T> は真に評価されますから、この条件式は必ず真 (T) になります。
つまり、上の 3 行がすべて不成立 (nil) でも、最後の条件式が必ず実行されます。

条件判断に基づいて実行の流れを変える制御関数としての Cond 関数は、次のようになります。

① if 関数は条件式が 1 つだけしか持てなかったのに対して、cond 関数は複数の条件式を並べることができます。つまり、複数の if 式を 1 つにまとめる方法です。
ある条件と、それが真のとき実行する式を組みにして何組でも並べることができます。

```
( cond
        (( 条件式 1)   (Then の処理 ))          ・・・条件式 1 が正しいときに実行する
        (( 条件式 2)   (Then の処理 ))          ・・・条件式 2 が正しいときに実行する
        (( 条件式 3)   (Then の処理 ))          ・・・条件式 3 が正しいときに実行する
        (   T          (Then の処理 ))          ・・・上記以外のときに実行する
);cond
```

② cond 関数は引数に複数の式を並べられますから、条件に基づいて複数の式を実行する時でも progn を使わなくて済みます。

③ P1-126 の if 式を cond 式で記述すると、以下のようになります。

```
(setq  A  (getint "¥n1 か 2 の数値を入力してください。" ))
  (cond
    (( = A 1 ) (command "LINE" ))        ;A が <1> であれば、線分コマンドを実行
    (( = A 2 ) (command "CIRCLE" ))      ;A が <2> であれば、円コマンドを実行
    (    T     (command "RECTANG" )) ;<1> でも <2> でもなければ、長方形コマンドを実行
  );cond
```


ChgCircle.lsp	選択した円の半径を変更する
目的：	選択した円の半径が <10> のときは <5> に、<20> のときは <30> に変更します。
主要関数：	<entsel><car><entget><assoc><cdr><cons><subst><entmod>

Step1 ー 円を選択します。

① (setq ent1 (entget (car (entsel "¥n 円を指示してください。")))) ❶
円の図形名を変数 <ent1> にセットします。
((-1 . < 図形名 : 7ef03958>) (0 . "CIRCLE") (330 . < 図形名 : 7ef01cf8>) (5 . "29B")
(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle")
(10 1074.29 1475.39 0.0) (40 . 10.0) (210 0.0 0.0 1.0))

②図形名 ent1 の図形タイプ（LINE、CIRCLE 等）を取得し、ent2 にセットします。❷
ent2 (cdr (assoc 0 ent1))

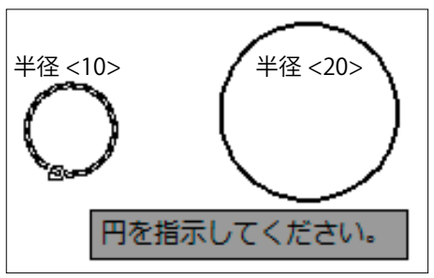
③ (if (/= ent2 "CIRCLE") ❸❹❺❻❼❽
選択した図形が円かどうか判断します。
→ 円でなければ、" 選択した図形は円ではありません。" を表示して終了します。
→ 円の時、半径が <10> か <20> によってプログラムが分かれます。

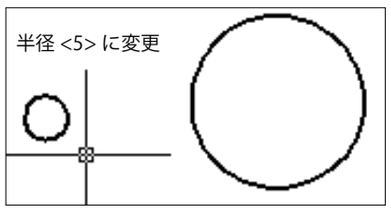
Step2 ー ((= rad 10.0) (setq ent1 (subst new_rad_5 old_rad ent1))(entmod ent1)) ❾❿
①円の半径が <10> であれば、その半径を <5> に変更します。
② (entmod ent1) でデータを更新します。（必須です）

Step3 ー ((= rad 20.0) (setq ent1 (subst new_rad_30 old_rad ent1))(entmod ent1)) ⓫⓬
①円の半径が <20> であれば、その半径を <30> に変更します。
② (entmod ent1) でデータを更新します。（必須です）

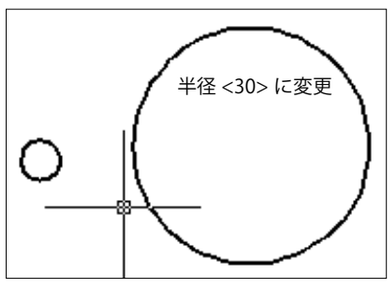
Step1 ー半径 <10> の円を選択したとき。（左側）

元の図形です。





Step1 ー半径 <20> の円を選択したとき。（右側）



```
(defun c:ChgCircle( / ent1 ent2 old_rad new_rad rad)
❶  (setq ent1 (entget (car (entsel "¥n 円を指示してください。"))))
❷  (setq ent2 (cdr (assoc 0 ent1)))  ;ent1 の図形タイプ（LINE、CIRCLE 等）を取得
❸  (if (/= ent2 "CIRCLE")  ; 円でなければ
❹      (prompt "¥n 選択した図形は円ではありません。")
       (progn  ;円であれば、以下を実行する
❺      (setq old_rad (assoc 40 ent1))  ;(40 . 半径 ) のドット・ペアを取得
❻      (setq new_rad_5 (cons 40 5.0))  ; あらかじめ、半径が <5> のドット・ペアを作成
❼      (setq new_rad_30 (cons 40 30.0))  ; あらかじめ、半径が <30> のドット・ペアを作成
❽      (setq rad (cdr old_rad))  選択した円の半径を変数 <rad> にセット
       (cond  ;rad が <10.0> の時は 1 番目の処理を行い、<20.0> の時は 2 番目の処理を行う
❾        ((= rad 10.0) (setq ent1 (subst new_rad_5 old_rad ent1))
❿                      (entmod ent1))
⓫        ((= rad 20.0) (setq ent1 (subst new_rad_30 old_rad ent1))
⓬                      (entmod ent1))
       );cond
    );progn
  );if
  (princ)
) ;end
```

→ Step1

→ Step2

→ Step3

番号	関数名	説　明
①	ent1	entsel で選択した図形の情報を取得します。ent1 には図形名と座標がセットされます。
②	ent2	図形 ent1 の図形タイプは グループコード <0> にあります。 cdr 関数で図形のタイプ (線分とか円とか) の情報が得られます。
③	(/= ent2 "CIRCLE")	ent2 が円かどうかを判定します。
④	prompt	選択した図形が円でなければ、このメッセージを表示して終了します。 円であれば、progn 以下に進みます。
⑤	old_rad	円の半径を取得します。
⑥	new_rad_5	半径のグループコード <40> と半径 <5> を組み合わせて、ドットペアをあらかじめ作成します。
⑦	new_rad_30	半径のグループコード <40> と半径 <30> を組み合わせて、ドットペアをあらかじめ作成します。
⑧	rad (cdr old_rad)	選択した円の半径を取得して、半径が <10> か <20> の判断をします。
⑨	subst	半径 <10> と半径 <5> を入れ替えます。
⑩	(entmod ent1)	データを更新します。（必須です）
⑪	subst	半径 <20> と半径 <30> を入れ替えます。
⑫	(entmod ent1)	データを更新します。（必須です）

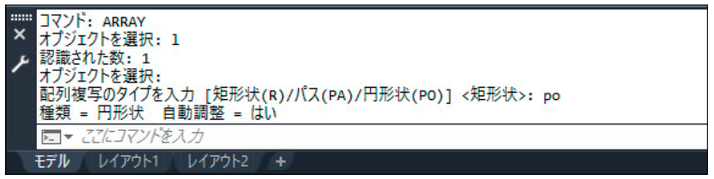
第 3 章　AutoCAD と対話する

Word や Excel ではプログラム側からユーザー側に問い掛けてくることはありません。しかし、AutoCAD ではお互いに会話しながら作業を進めます。
大事なことは相手側 (AutoCAD) からの問い掛けに正確に答えることです。
では、AutoCAD はどのように問い掛けてくるのでしょうか？

この章では以下のことを学びます。

第 1 節　表示をコントロールする

AutoCAD が問い掛けてくる窓口がコマンド ウィンドウです。（またはコマンド ライン）
このウィンドウを通して AutoCAD とユーザーは会話します。



[status(図面情報)] コマンドのように情報量が多いときは、[F2] キーか [textscr] コマンドで
テキスト ウィンドウに切り替えます。

使用頻度の高い AutoLISP の 表示コントロール関数を、次の表に示します。（順不同）

①メッセージ表示コントロール関数	
関　数	説　明
(prin1 [expr [file-desc]])	コマンド ラインに式を表示したり、開いているファイルに式を書き出します。
(princ [expr [file-desc]])	コマンド ラインに式を表示したり、開いているファイルに式を書き出します。
(print [expr [file-desc]])	コマンド ラインに式を表示したり、開いているファイルに式を書き出します。
(prompt msg)	スクリーンのコマンド ラインに文字列を表示します。
(terpri)	コマンドラインに改行を出力します。

②画面の表示コントロール関数	
関　数	説　明
(graphscr)	AutoCAD のグラフィックス スクリーンを表示します。
(textpage)	グラフィックス スクリーンからテキスト スクリーンに切り替えます。
(textscr)	グラフィックス スクリーンからテキスト スクリーンに切り替えます。(AutoCAD のスクリーン切り替えファンクション キーと同じ)
(redraw [ename [mode]])	現在のビューポートまたは現在のビューポートで指定されたオブジェクト (図形) を再描画します。

③コマンド ラインへの表示をコントロールするシステム変数	
関　数	説　明
CMDECHO	command 関数から実行されたコマンドの表示 / 非表示をコントロールします。

④ダイアログの表示を非表示にできる代表的なコマンド	
関　数	説　明
-hatch	コマンドの前に <-> を付けると、[ハッチング] ダイアログを表示させない。
-insert	コマンドの前に <-> を付けると、[ブロック挿入] ダイアログを表示させない。
-view	コマンドの前に <-> を付けると、[ビュー管理] のダイアログを表示させない。
-layer	コマンドの前に <-> を付けると、[画層プロパティ管理] ダイアログを表示させない。
-color	コマンドの前に <-> を付けると、[色] ダイアログを表示させない。
-linetype	コマンドの前に <-> を付けると、[線種管理] ダイアログを表示させない。
-lweight	コマンドの前に <-> を付けると、[線の太さの設定] のダイアログを表示させない。
-style	コマンドの前に <-> を付けると、[文字スタイル管理] ダイアログを表示させない。
-dimstyle	コマンドの前に <-> を付けると、[寸法スタイル管理] ダイアログを表示させない。

1 メッセージ表示コントロール関数

PRIN1	コマンド ラインに式を表示したり、開いているファイルに式を書き出します。

① (prin1 "AutoLISP")　→ "AutoLISP""AutoLISP"
② (prin1 "AutoLISP")(princ)　→ "AutoLISP"

→① prin1 はクォーテーションで囲まれた文字列を表示し、戻り値もクォーテーションで囲まれた文字列を表示します。
→② (princ) を追加すると、クォーテーションで囲まれた文字列だけを表示します。

PRINC	コマンド ラインに式を表示したり、開いているファイルに式を書き出します。

① (princ "AutoLISP")　→ AutoLISP"AutoLISP"
② (princ "AutoLISP")(princ)　→ AutoLISP

→① princ はクォーテーションで囲まれない文字列を表示し、戻り値はクォーテーションで囲まれた文字列を表示します。
→② (princ) を追加すると、文字列だけを表示します。

PROMPT	コマンド ラインに文字列を表示します。

① (prompt "AutoLISP")　→ AutoLISPnil
② (prompt "AutoLISP")(princ)　→ AutoLISP

→① prompt はクォーテーションで囲まれない文字列を表示し、戻り値は <nil> を返します。
→② (princ) を追加すると、文字列だけを表示します。

TERPRI	文字列を改行してコマンド ラインに表示します。

①コマンド : (prompt "AutoLISP")　→ AutoLISPnil
②コマンド : (prompt "AutoLISP")(terpri)　→ AutoLISP
nil （← nil が改行されて、次の行に表示される。）

(例)
(prompt "1 点目を指示：") (terpri) (prompt "2 点目を指示：")
コマンドラインには、次のように表示されます。
1 点目を指示：
2 点目を指示：

面取り（prompt と princ の組み合わせ）

Step1 ー プログラムの最初に、面取りのシステム変数 <CHAMMODE> を <0> に設定します。❶
<CHAMMODE> は面取りを作成するときの入力方法を設定します。
<0> は 2 つの面取り距離を指定し、<1> は面取り距離と角度を指定します。
このプログラムは 2 本線の面取りですから、<CHAMMODE> を <0> にします。
次に、現在の面取りのシステム変数 <CHAMFERA> と <CHAMFERB> を取得します。❷❸
最後に元の設定に戻すためです。

```
(setq cha_a (getvar "CHAMFERA")) (setq cha_b (getvar "CHAMFERB"))
```

Step2 ー 1 本目の面取りの距離を求めます。❹❺❻❼

```
(setq dis 10.0)
(prompt "¥n1 本目の面取りの距離は？ <")(princ dis)
(setq dis1 (getreal ">"))
(if (= dis1 nil) (setq dis1 dis))
```

以下がコマンドラインに表示されます。
→1 本目の面取りの距離は？ <10.0>
<20> と入力します。
もし、空打ち時は、dis の数値が dis1 に代入されます。

Step3 ー 2 本目の面取りの距離を求めます。❽❾❿

```
(prompt "¥n2 本目の面取りの距離は？ <")(princ dis1)
(setq dis2 (getreal ">"))
(if (= dis2 nil) (setq dis2 dis1))
```

以下がコマンドラインに表示されます。
→2 本目の面取りの距離は？ <20.0>
もし、空打ち時は、dis1 の数値が dis2 に代入されます。

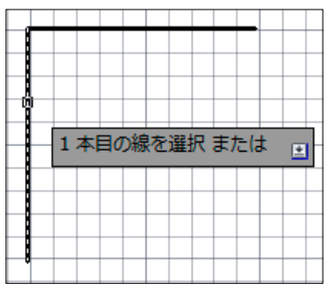
Step4 ー面取りコマンドを使って、2 本の線分を面取りします。⓫⓬⓭⓮

```
(setq en1 (car (entsel "¥n1 本目の線分を選択：")))   → 面取りの 1 本目を指示します。
(setq en2 (car (entsel "¥n2 本目の線分を選択：")))   → 面取りの 2 本目を指示します。
```

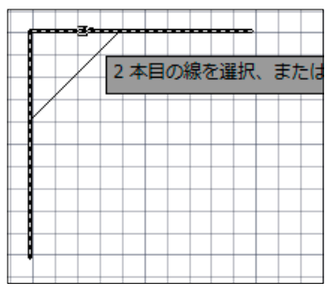
Step5 ー最後にシステム変数 <CHAMFERA> と <CHAMFERB> を元に戻します。⓯⓰

```
(setvar "CHAMFERA" cha_a) (setvar "CHAMFERB" cha_b)
```

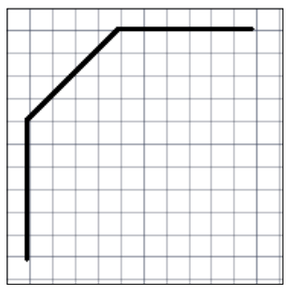
Step4 ー 1 本目の線分を選択



Step4 ー 2 本目の線分を選択



結果




```
(defun c:trim_1( / cha_a cha_b dis dis1 en1 en2)
❶ (setvar "CHAMMODE" 0) ; 面取りモードを <0> にセットする
❷ (setq cha_a (getvar "CHAMFERA")) ;1 本目の面取りの初期値を取得する
❸ (setq cha_b (getvar "CHAMFERB")) ;2 本目の面取りの初期値を取得する
                                                              → Step1
❹ (setq dis 10.0) ; 面取りの初期値を <10.0> にセットする
❺ (prompt "¥n1 本目の面取りの距離は？ <")(princ dis) ; メッセージを表示する
❻ (setq dis1 (getreal ">")) ;1 本目の面取りの距離を取得する
❼ (if (= dis1 nil) (setq dis1 dis)) ; 空打ちの時は、<dis> の値を <dis1> にセットする
                                                              → Step2
;2 本目の面取りの距離
❽ (prompt "¥n2 本目の面取りの距離は？ <")(princ dis1) ; メッセージを表示する
❾ (setq dis2 (getreal ">")) ;2 本目の面取りの距離を取得する
❿ (if (= dis2 nil) (setq dis2 dis1)) ; 空打ちの時は、<dis1> の値を <dis2> にセットする
                                                              → Step3
;2 本の線分を指示して面取りする図形を取得する
⓫ (setq en1 (car (entsel "¥n1 本目の線分を選択："))) ;1 本目の線分を取得する
⓬ (setq en2 (car (entsel "¥n2 本目の線分を選択："))) ;2 本目の線分を取得する
                                                              → Step4
; 面取りコマンドを実行 する
⓭ (command "chamfer" "d" dis1 dis2 "") ; 面取り距離をセットする
⓮ (command "chamfer" en1 en2) ; 面取りコマンドを実行する
⓯ (setvar "CHAMFERA" cha_a) ;1 本目の面取り距離を元に戻す
⓰ (setvar "CHAMFERB" cha_b) ;2 本目の面取り距離を元に戻す
                                                              → Step5
(princ)
) ;end
```

番号	関数名	説　明
①	"CHAMMODE" 0	CHAMMODE <0> は、2 本の面取り。<1> は線分と角度の面取り。
②	getvar "CHAMFERA"	1 本目の面取りの距離を取得します。最後に元に戻すため。
③	getvar "CHAMFERB"	2 本目の面取りの距離を取得します。最後に元に戻すため。
④	dis 10.0	面取りの距離の初期値に <10.0> をセットします。
⑤	<")(princ dis)	コマンドラインに、"<10.0" と表示されます。
⑥	(getreal ">"	コマンドラインに、">" が表示されます。結果として、"<10.0>"。
⑦	(= dis1 nil) (setq dis1 dis)	ユーザーが空打ちした場合は、変数 [dis1] に dis の値をセット。
⑧	<")(princ dis1)	コマンドラインに、"<dis1" の値が表示されます。
⑨	(getreal ">")	コマンドラインに、">" が表示されます。結果として、"<dis1>"。
⑩	(= dis2 nil) (setq dis2 dis1)	ユーザーが空打ちした場合は、変数 [dis2] に dis1 の値をセット。
⑪	en1	選択した線分の図形名を取得します。面取り時に指定します。
⑫	en2	選択した線分の図形名を取得します。面取り時に指定します。
⑬	"chamfer" "d" dis1 dis2 ""	面取りコマンドの、1 本目と 2 本目の距離を指定します。
⑭	"chamfer" en1 en2	面取りコマンドで 2 本の線分を指定します。
⑮	"CHAMFERA" cha_a	1 本目の面取りの距離を元に戻します。
⑯	"CHAMFERB" cha_b	2 本目の面取りの距離を元に戻します。

entsel は、P1-210 を参考

2 画面の表示コントロール関数

TEXTSCR	文字画面に切り替えます。

作図画面から文字画面に切り替える　→ (textscr)

GRAPHSCR	作図画面に切り替えます。

文字画面から作図画面に切り替える　→ (graphscr)

REDRAW	現在のビューポートを再描画します。

(redraw ent1 mode))　→ ent1 は図形名

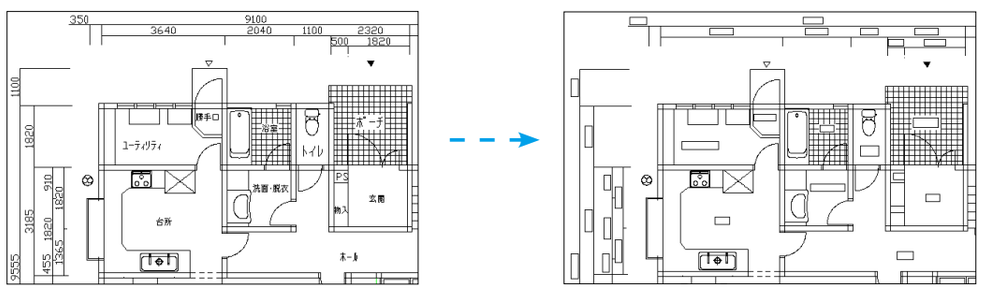
mode 1 = 図形の再描画
2 = 図形の消去
3 = 図形のハイライト表示
4 = 図形のハイライト解除

(redraw)　→ redraw コマンドと同じです。
(redraw ent1 1)　→ 図形を再描画します。
(redraw ent1 2)　→ 図形を消去します。
(redraw ent1 3)　→ 図形をハイライト表示します。
(redraw ent1 4)　→ 図形のハイライトを解除します。

QTEXT	文字と属性の表示と印刷をコントロールします。

① [ツール] → [オプション] → [表示] タブから [文字の境界フレームのみを表示] を選択
②コマンド ラインから、[QTEXT] → [ON/OFF] → [REGEN] と入力

[QTEXT] コマンドで文字省略モードをオンにすると、図面内の文字や属性文字が文字を囲む境界ボックスとして表示されます。これによって、図面内に多くの文字が有る時は再描画と再作図にかかる時間が短縮できます。

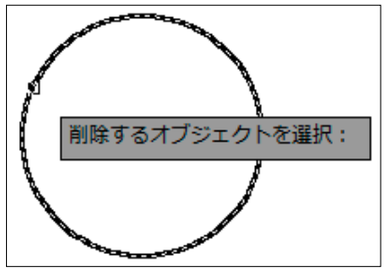


redraw 関数	削除する図形をハイライト表示する

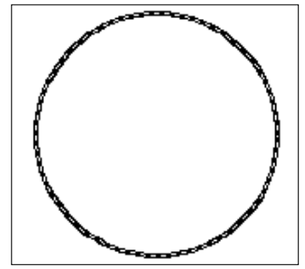
```
❶ (setq ent1 (entsel "¥n 削除するオブジェクトを選択 : "))
❷ (setq ent2 (car ent1)) ; 選択した図形名を変数 <ent2> にセット
❸ (redraw ent2 3) ; 図形 ent2 をハイライト表示する
❹ (Initget 1 "Yes No") ; 次の行のキーを "Y" と "N" に固定
❺ (setq ans (getkword " この図形でよろしいですか ? Yes/No : "))
❻ (if (or (= ans "Y") (ans = "y")) ;"Y" か "y" のキーが押されたら
❼ (redraw ent2 2) ; 図形を削除
❽ (redraw ent2 4) ; ハイライト表示を解除
);if
```

→ Step1
→ Step2
→ Step3
→ Step4

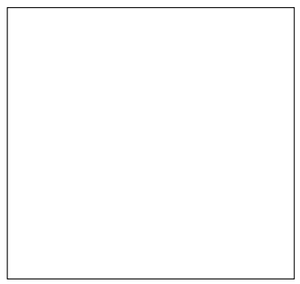
Step1: オブジェクトを選択



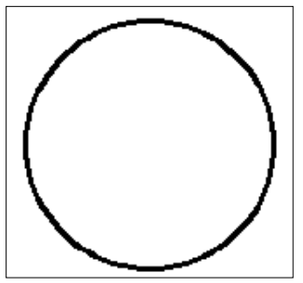
Step2: 選択したオブジェクトをハイライト表示



Step3: getkword で "Y" ならば、削除



Step4: getkword で "N" ならば、ハイライトを解除



番号	関数名	説　明
①	entsel	削除するオブジェクトを選択します。(図形名と指示した座標値を取得します。)
②	car ent1	car 関数で図形名だけを取得し、変数 ent2 にセットします。
③	redraw 3	図形 ent2 をハイライト表示します。
④	initget 1	0(ゼロ) を入力することを禁止します。
⑤	getkword	キーボード入力は、"Y" か "N" に限定します。
⑥	if	キーボードからの入力が、"Y" か "y" の時の処理を記述します。
⑦	redraw 2	"Y" ならば、削除します。
⑧	redraw 4	"N" ならば、ハイライトの表示を解除します。

☞initget、getkword は、P1-166、P1-167 を参考

AutoCAD と対話する

3 コマンド ラインへの表示をコントロールするシステム変数

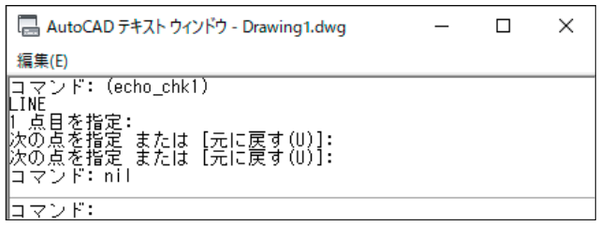
CMDECHO	command 関数から実行されたコマンドの表示 / 非表示を切り替えます。

AutoLISP の command 関数を実行中に、プロンプトとユーザー入力をエコーバック表示するかをコントロールします。

(cmdecho 0) → エコーバックを表示しません。
(cmdecho 1) → エコーバックを表示します。

例① システム変数 <CMDECHO> が <0> の場合。

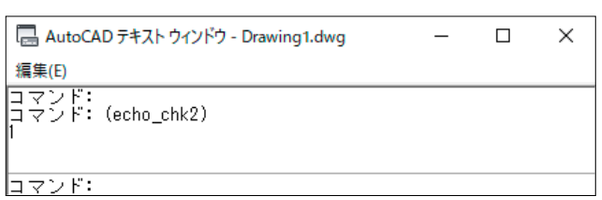
```
(defun echo_chk1( )
   (setq pt1 (list 100 100))                  →  変数 pt1 に XY 座標 (100,100) をセット
   (setq pt2 (list 200 200))                  →  変数 pt2 に XY 座標 (200,200) をセット
   (command "LINE" pt1 pt2 "")                →  pt1 から pt2 に線分を作図する
)
```



LINE コマンドのメッセージが表示されます。

例② 次に、システム変数 <CMDECHO> を <1> に変更します。

```
(defun echo_chk 2( )
   (setvar "cmdecho" 0)                       →  コマンド ラインにメッセージを表示しない
   (setq pt1 (list 100 100))
   (setq pt2 (list 200 200))
   (command "LINE" pt1 pt2 "")
   (setvar "cmdecho" 1)                       →コマンド ラインにメッセージを表示させる
)
```



LINE コマンドのメッセージが表示されません。

最後の <1> は cmdecho が <1> にセットされたことを表示しています。
この表示をしないようにするには、最後に (princ) を追加します。

4 ダイアログの表示を非表示にできる代表的なコマンド

(例) -hatch、-insert	ハッチングやブロック挿入のダイアログを表示させません

AutoCAD には、ダイアログを表示するコマンドがあります。通常、このダイアログはユーザーに処理を判りやすくする機能ですが、AutoLISP で使用する場合は問題があります。

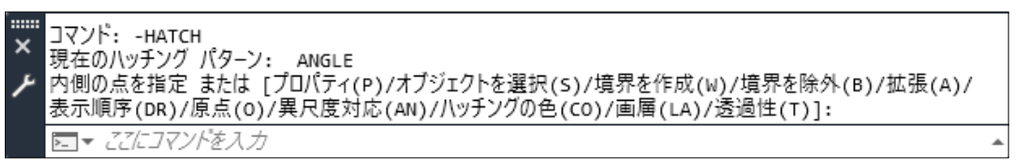
それは、LISP の途中でダイアログが表示されると、ユーザーの選択待ちの状態になるために、LISP プログラムが停止してしまうからです。

このトラブルを避けるために、コマンドの前に < ー > 記号を付加するとダイアログが非表示になり、LISP プログラムが最後まで動作します。

-hatch	ハッチングのダイアログを表示させない

普通に <hatch(T)> コマンドを使った場合、ダイアログが表示されます。

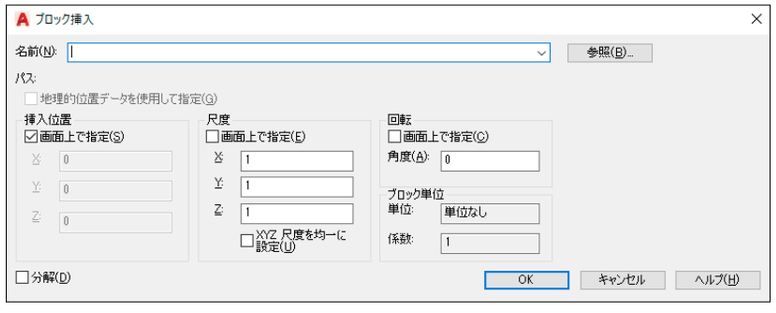
hatch の前に < ー > 記号を付加して <-hatch> とすると、ダイアログを表示しません。



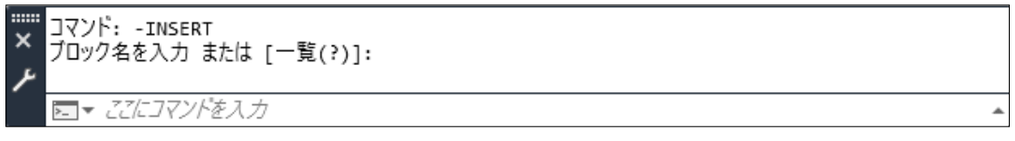
コマンド : -hatch
現在のハッチング パターン : ANGLE
内側の点を指定 または [プロパティ (P)/ オブジェクトを選択 (S)/ 境界を作成 (W)/ 境界を除外 (B)/ 拡張 (A)/ 表示順序 (DR)/ 原点 (O)/ 異尺度対応 (AN)/ ハッチングの色 (CO)/ 画層 (LA)/ 透過性 (T)]:

-insert	ブロック挿入のダイアログを表示させない

普通に <ClassicInsert> コマンドを使った場合（ダイアログが表示されます。）



insert の前に < ー > 記号を付加して <-insert> とすると、ダイアログを表示しません。

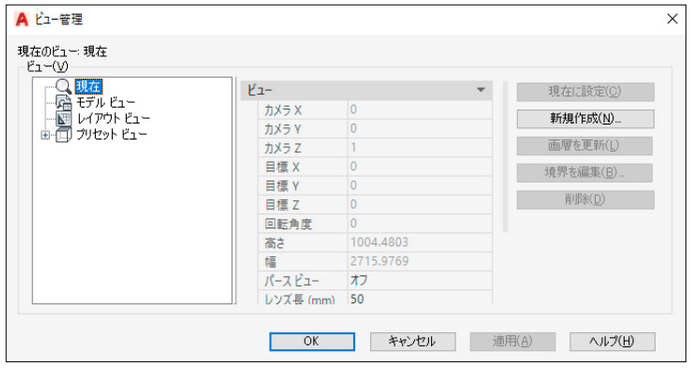


コマンド : -INSERT

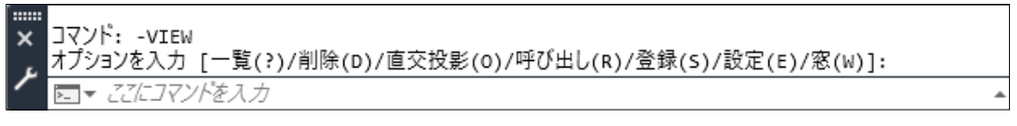
ブロック名を入力 または [一覧 (?)]:

-view	ビュー管理のダイアログを表示させない

普通に <view> コマンドを使った場合（ダイアログが表示されます。）



view の前に < ー > 記号を付加して <-view> とすると、ダイアログを表示しません。

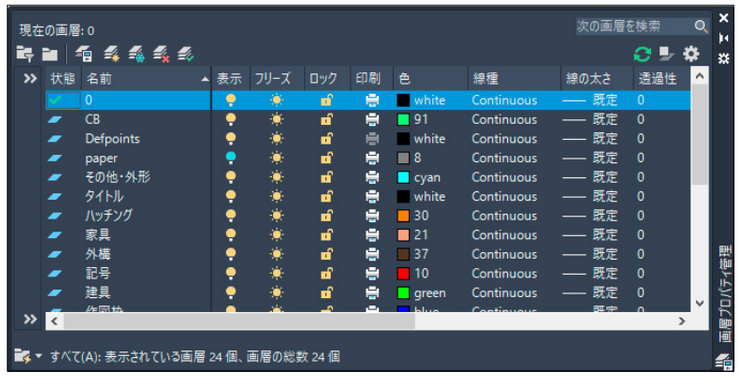


コマンド : -VIEW

オプションを入力 [一覧 (?)/ 削除 (D)/ 直交投影 (O)/ 呼び出し (R)/ 登録 (S)/ 設定 (E)/ 窓 (W)]:

-layer	画層のダイアログを表示させない

普通に <layer> コマンドを使った場合（ダイアログが表示されます。）



layer の前に < ー > 記号を付加して <-layer> とすると、ダイアログを表示しません。

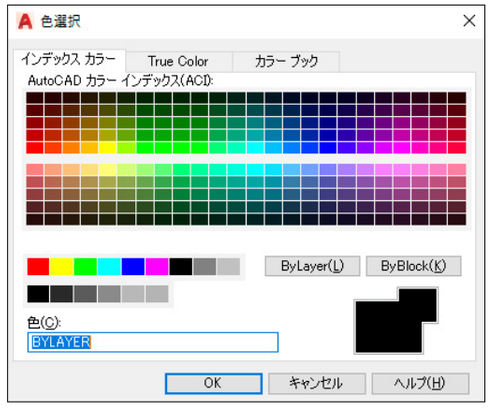
コマンド : -LAYER

現在の画層 : "0"

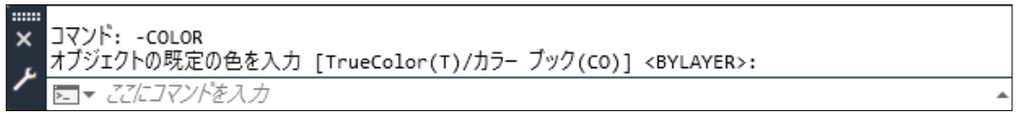
オプションを入力 [一覧 (?)/ 現在の層の新規作成 (M)/ 現在の層の変更 (S)/ 新規作成 (N)/ 名前変更 (R)/ 表示 (ON)/ 非表示 (OFF)/ 色設定 (C)/ 線種設定 (L)/ 線の太さ (LW)/ 透過性 (TR)/ マテリアル (MAT)/ 印刷 (P)/ フリーズ (F)/ フリーズ解除 (T)/ ロック (LO)/ ロック解除 (U)/ 画層状態 (A)/ 説明 (D)/ 正規画層 (E)/ 外部参照 (X)]:

-color	色変更のダイアログを表示させない

普通に <color> コマンドを使った場合（ダイアログが表示されます。）



color の前に < ー > 記号を付加して <-color> とすると、ダイアログを表示しません。

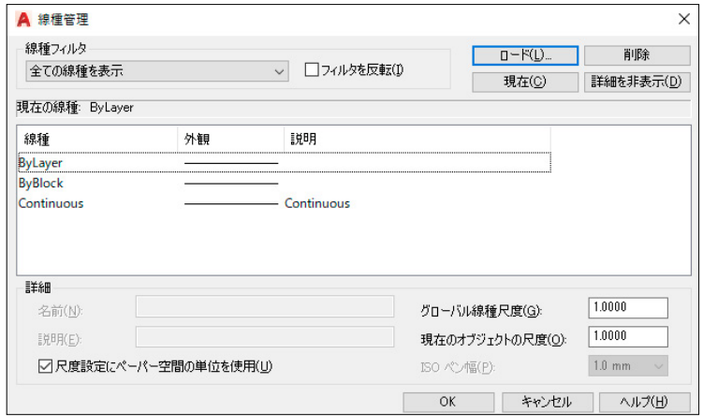


コマンド : -color

オブジェクトの既定の色を入力 [TrueColor(T)/ カラー ブック (CO)] <BYLAYER>:

-linetype	線種のダイアログを表示させない

普通に <linetype> コマンドを使った場合（ダイアログが表示されます。）

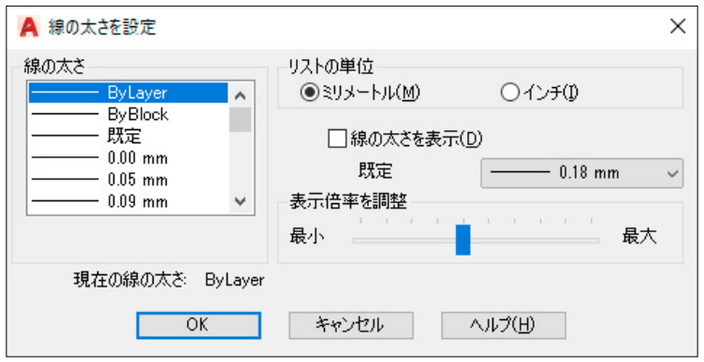


linetype の前に < ー > 記号を付加して <-linetype> とすると、ダイアログを表示しません。

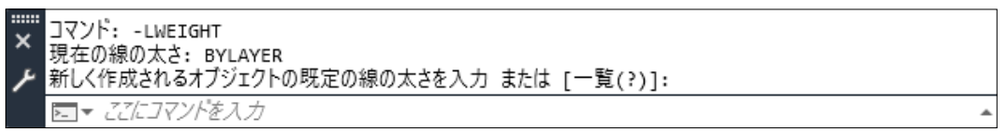
コマンド : -LINETYPE
現在の線種 : "ByLayer"
オプションを入力 [一覧 (?)/ 作成 (C)/ ロード (L)/ 設定 (S)]:

-lweight	線の太さの設定のダイアログを表示させない

普通に <lweight> コマンドを使った場合（ダイアログが表示されます。）



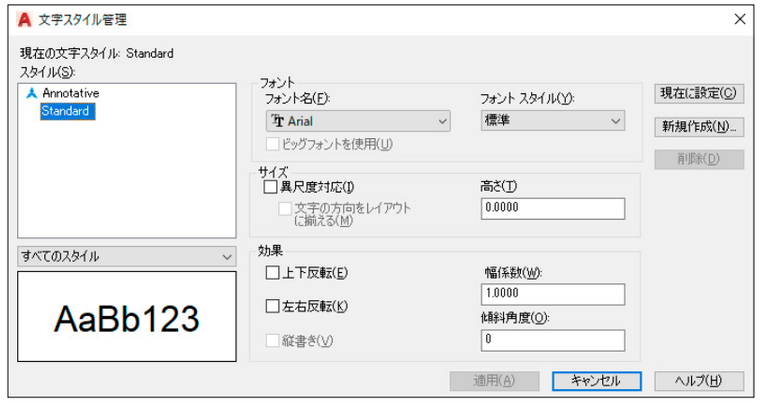
lweight の前に < ー > 記号を付加して <-lweight> とすると、ダイアログを表示しません。



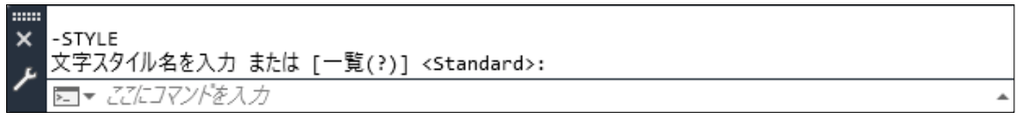
コマンド : -LWEIGHT
現在の線の太さ : BYLAYER
新しく作成されるオブジェクトの既定の線の太さを入力 または [一覧 (?)]:

-style	文字スタイルのダイアログを表示させない

普通に <style> コマンドを使った場合（ダイアログが表示されます。）



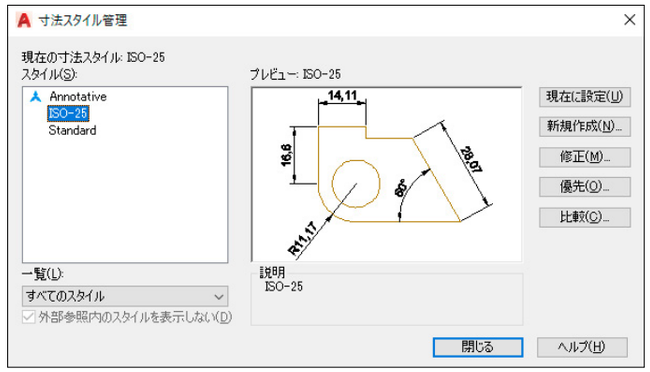
style の前に < ー > 記号を付加して <-style> とすると、ダイアログを表示しません。



コマンド :-STYLE
文字スタイル名を入力 または [一覧 (?)] <Standard>:

-dimstyle	寸法スタイルのダイアログを表示させない

普通に <dimstyle> コマンドを使った場合（ダイアログが表示されます。）



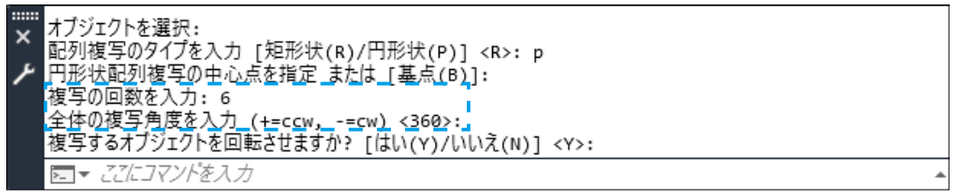
dimstyle の前に < ー > 記号を付加して <-dimstyle> とすると、ダイアログを表示しません。

コマンド : -DIMSTYLE
現在の寸法スタイル : ISO-25　異尺度対応 : いいえ
寸法スタイル編集オプションを入力
[異尺度対応 (AN)/ 登録 (S)/ 呼び出し (R)/ 現在のスタイル変数一覧 (ST)/ スタイル変数一覧 (V)/
適用 (A)/ 一覧 (?)] < 呼び出し >:

AutoCAD と対話する

第 2 節　ユーザー入力関数

AutoCAD 側からの問い掛けに対して、ユーザーはキーボードやマウスで応答します。
キーボードから入力できるのは、整数と実数、文字列です。一方マウスからは XYZ の座標値が入力できます。
しかし、入力するタイプによって使用される関数は違います。下図は、[array(回転複写)] におけるコマンド ラインのメッセージですが、上から [回数]、[角度] を尋ねています。
それぞれの問いに対応する数値を入力しなければ、求める図形が作図できません。



この節では、このようなユーザー入力のタイプごとに異なる関数を学びます。

使用頻度の高い AutoLISP のユーザー入力関数を、次の表に示します。（順不同）

①ユーザー入力関数（キーボード入力）	
関　数	説　明
(getint [msg])	ユーザーが整数を入力するまで待機 (一時停止) し、入力された整数を返します。
(getreal [msg])	ユーザーが実数を入力するまで待機 (一時停止) し、入力された実数を返します。
(getstring [cr] [msg])	ユーザーが文字列を入力するまで待機 (一時停止) し、入力された文字列を返します。

②ユーザー入力関数 (マウス & キーボード入力)	
関　数	説　明
(getangle [pt] [msg])	ユーザーが角度を入力するまで待機 (一時停止) し、入力された角度をラジアン単位で返します。
(getcorner pt [msg])	ユーザーが長方形の 2 番目のコーナーを入力するまで待機 (一時停止) します。
(getdist [pt] [msg])	ユーザーが距離を入力するまで待機 (一時停止) します。
(getorient [pt] [msg])	ユーザーが角度を入力するまで待機 (一時停止) し、入力された角度をラジアン単位で返します。
(getpoint [pt] [msg])	ユーザーが点を入力するまで待機 (一時停止) し、入力された点を返します。

③ユーザー入力関数（キーワード）	
関　数	説　明
(getkword [msg])	ユーザーがキーワードを入力するまで待機 (一時停止) し、入力されたキーワードを返します。
(initget [bits] [string])	次のユーザー入力関数の呼び出しで使用するキーワードを設定します。

1　ユーザー入力関数（キーボード入力）

GETINT	整数を入力します。

(getint "¥n 複写回数を入力："))

もし実数を入力すると、< 整数値を入力してください。> のメッセージが表示されます。

GETREAL	実数を入力します。

(getreal "¥n 数値を入力："))

もし整数 <3> を入力すると、<3.0>（実数）が返されます。

GETSTRING	文字列を入力します。

(setq text1 (getstring "¥n 文字を入力："))

コマンド : AutoLISP　　→ コマンドラインに < AutoLISP > と入力します。
コマンド : !text1　　→ コマンドラインに < !text1 > と入力します。
"AutoLISP"　　→ "AutoLISP" が表示されます。

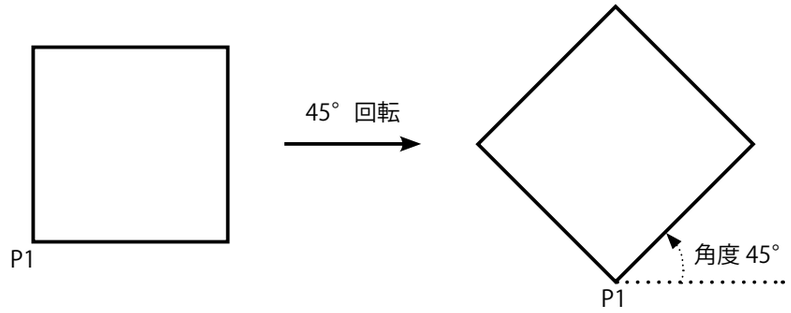
Point!

Lisp の記述	コマンド ラインに表示される文字列
(getint "¥n 複写回数を入力："))	→ < 複写回数を入力：> が表示されます。 ユーザーはこの指示を確認して、入力します。
(getint)	→何もメッセージは表示されず、ユーザーの 入力待ちになります。

Point!

(setq text1 (getstring T "¥n 文字を入力："))
上記のように、getstring の次に <T> などの英数字を付加すると（つまり nil ではない）、空白の文字を受け付けます。
もし getstring だけの場合は、空白のスペースキーを押したとき、入力は終了します。

Rota1.lsp	四角形を回転する
目的：	回転コマンドを使いマウスで点 P1 を指示した後、変数に代入されている数値で図形を回転させます。
主要関数：	<getreal><getpoint><ROTATE>



①最初に、回転角度を入力させ、変数 ang1 にセットします。
②次に回転の基点の座標をマウスで指示させます。(P1)
③次に回転コマンドを使います。
④ユーザーに回転する図形 (四角形) を選択させますが、その後の回転の基点と回転角度は先に入力済みですから、プログラムでその数値を使って四角形を回転させます。

```
❶ (defun C:Rota1( / ang1 p1)                                  → 関数を宣言します。
❷   (setq ang1 (getreal "¥n 回転角度 : " ))                    → ユーザーに、回転角度と基点を求めます。
❸   (setq p1 (getpoint "¥n 回転基点 : " ))                     → ユーザーに、回転角度と基点を求めます。
   ; 回転角度と基点が確定したので、実行する
❹   (command "ROTATE" pause "" p1 ang1)                       → ユーザーに、オブジェクトを選択させます。
) ;end
```

番号	関数名	説　明
①	defun	関数の宣言です。 Rota1 で宣言する場合は、(defun C:Rota1(/) になります。(/ ang1 p1) はこのプログラムの中だけで使用できるローカル変数です。
②	getreal	ユーザーから入力された数値を実数として認識します。
③	getpoint	ユーザーがマウスで指示した座標を変数 p1 にセットします。
④	"ROTATE"	回転コマンドを使用します。 pause はユーザーの指示待ちです。(図形を指示するまで、プログラムが停止します。"" は選択し終わったという意味になります。)

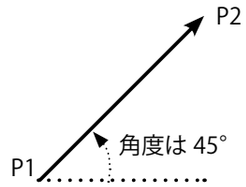
回転 (ROTATE) コマンドは、図形を指示し終わると、回転の基点を聞いてきます。
そして、次に回転角度を聞いてきます。

②は [getangle] 関数ではなく、[getreal] 関数であることに注意してください。

2 ユーザー入力関数（マウス & キーボード入力）

GETANGLE	指示した 2 点間の角度を返します。（ラジアン）

(setq ang1 (getangle "¥n 角度を測定する 2 点を指示 : "))
マウスで点 P1 と点 P2 を指示します。

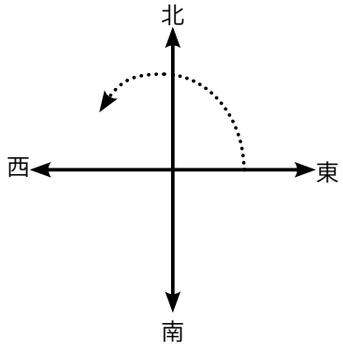


コマンド : !ang1　　→ コマンドラインに < !ang1 > と入力します。
0.785398　　→ ラジアンの値が表示されます。

getorient	東（右）の方向を 0° とした反時計回りの角度を返します。（ラジアン）

getangle 関数は、システム変数 <ANGDIR> と <ANGBASE> の影響を受けますが、getorient 関数は受けません。
必ず、東（右方向）が 0（ゼロ）ラジアンで反時計回りに計測します。

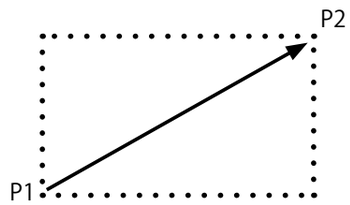
getangle 関数と getorient 関数は次のように使い分けをします。
getangle 関数・・他の座標や図形との相対角度を測るとき。
getorient 関数・・絶対角度を必要とするとき。（測量など）



GETCORNER	対角点を指示します。（基点からボックスが表示されます。）

getcorner で 1 点目を指示したあとに、ラバーバンドを表示さすには P1 を getcorner に続けます。
(getcorner P1 "¥n 対角点を指示 : "))

マウスで点 P1 と対角点 P2 を指示します。



AutoCAD と対話する

GETDIST	指示した 2 点間の距離を返します。

P1 と P2 の距離を変数 dist1 に代入します。
(setq dist1 (getdist "¥n2 点間を指示：")) → p1 と p2 を指示します。

コマンド : !dist1 → コマンドラインに < !dist1 > と入力します。
486.269 → 距離の値が表示されます。

GETPOINT	指示した点の座標値を返します。

(setq p1 (getpoint "¥n 挿入位置を指示："))

コマンド : !p1 → コマンドラインに < !p1 > と入力します。
(1301.34 839.554 0.0) → 指示した点の座標値が表示されます。

① Get 関数でプロンプト（メッセージ）を使うことができます。関数の後に 2 重引用符 (") で囲んでコマンドラインに表示する文字列を記述します。
（getpoint、getcorner、getangle、getdidt）
(setq P1 (getpoint "¥n1 点目を指示："))
¥n は改行キーです。次の行にプロンプトが表示されます。
このとき、コマンドラインに次のようなメッセージが表示されます。
1 点目を指示：

②次の点の座標 (P2) を得るときに、以下のように Getpoint の後に P1 をつけると、P1 からラバーバンドが表示され、視覚的に分かりやすくなります。
(setq P2 (getpoint P1 "¥n 終点を指示："))

この前の行で、(initget 32) を記述すると、ラバーバンドがハイライト（破線）表示に変わります。

getangle 関数	四角形を回転する

AutoCAD のコマンドに回転コマンドがありますが、このコマンドではユーザーに回転角度を聞いてきます。このときの関数が Getangle です。

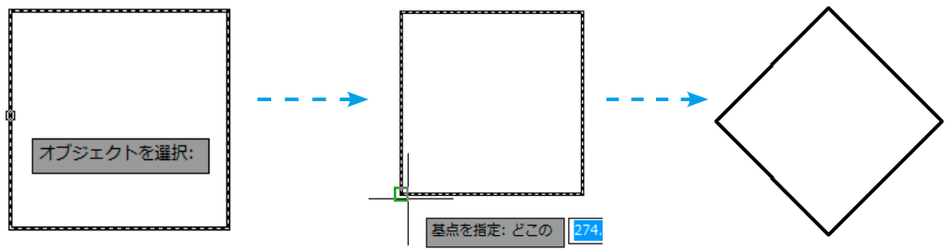
1. コマンドラインから、以下のように入力します。
(setq a (getangle))
この式は、コマンドラインを空白にして、角度の入力を待ちます。

2. キーボードから <45> と入力します。コマンドラインに <0.785398> 数字が表示されます。
これは、45 度をラジアンに変換した数値が返されています。
次に、このラジアンを度数に変換します。
180° が pi(3.14159) ラジアンですから、180 : pi = a : 0.785398 より、
a = (/ (* 0.7853 180.0) 3.14159) になります。

3. コマンドラインに、(setq a (/ (* 0.7853 180.0) 3.14159)) を入力します。
キーボードから、!a と入力すると、44.9944 の数値が表示されます。

4.[回転] コマンドを選び、図形を選択し基点を指示した後、< 回転角度 : > のメッセージが出たときに、キーボードから、!a と入力すると、基点を中心として図形が 45 度回転されます。

回転コマンドを使い、四角形を選択します。
回転の基点を左下のコーナーに指示します。
回転角度にキーボードから、!a と入力します。

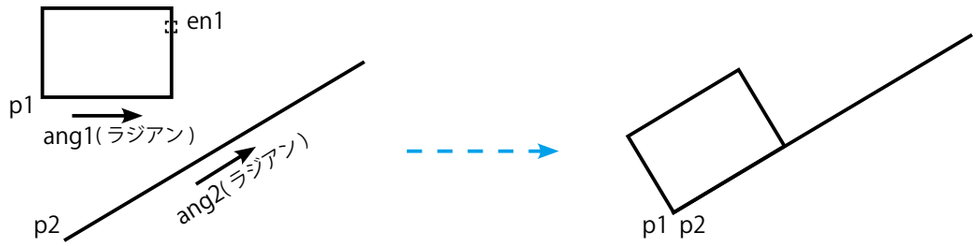


Point!

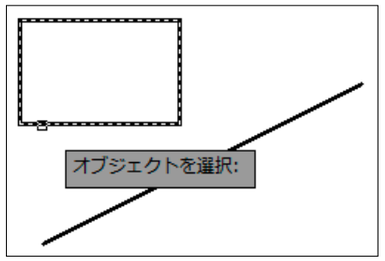
度数 (a) からラジアンへの変換式は
(* pi (/ a 180.0)) または、(/ (* pi a) 180.0)

ラジアン (a) から度数への変換式は
(* (/ a pi) 180.0) または、 (/ (* 180.0 a) pi)

MO_RO.lsp	選択した図形を移動して回転する
目的：	選択したオブジェクト（長方形）を線分に位置合わせします。（移動＋回転）
主要関数：	<entsel><car><getpoint><getangle><MOVE><ROTATE>



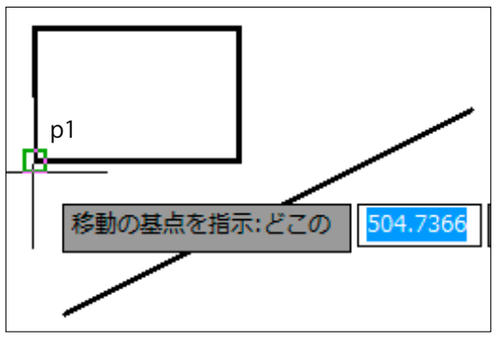
Step1 ー 回転移動するオブジェクト（長方形）を選択します。(entsel 関数) ❶❷



(setq en1 (entsel "¥n オブジェクトを選択："))
→ 回転移動するオブジェクトを選択します。
(< 図形名 : 7ef03958> (1350.56 1111.84 0.0)) が取得できます。

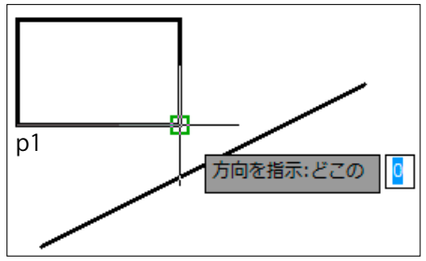
(setq en2 (car en1))
→ en1 から図形名だけ取り出します。
< 図形名 : 7ef03958> が取得できます。

Step2 ー getpoint 関数で移動する基点を指示し、変数 <p1> にセットします。❸



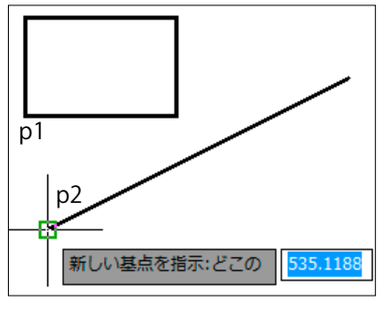
(setq p1 (getpoint "¥n 移動の基点を指示 :"))
→ 長方形の点 p1 を指示します。

Step3 ー getangle 関数で長方形を位置合わせする向きを決めます。❹



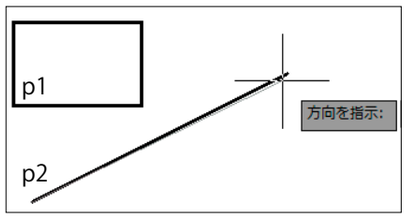
(setq ang1(getangle p1 "¥n 方向を指示 :"))
→ 長方形の片方の端を指示します。

Step4 ー 長方形の <p1> の移動先を決めます。<p2> ❺



(setq p2 (getpoint "¥n 新しい基点を指示 :"))
→ 点 p1 の移動先を指示します。

Step5 ～ 6 ー 長方形の移動先の向きを決めます。❻❼



(setq ang2(getangle p2 "¥n 方向を指示 :"))
→ 移動先での方向を指示します。

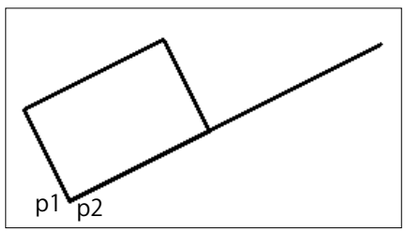
(/ (* (- ang2 ang1) 180.0) pi)
→ ラジアンを度数に変換します。

Point!

Step7 ー移動コマンドを使い、長方形の点 <p1> を点 <p2> に移動させます。❽

(command "MOVE" en2 "" p1 p2)
→ 長方形を基点 (p1) から目的点 (p2) へ移動します。

Step8 ー 回転コマンドを使い、長方形を点 <p2> を基点に回転させます。❾



(command "ROTATE" en2 "" p2 ang3)
→ 長方形を基点 (p2)、回転角度 (ang3) で回転します。

AutoCAD と対話する

```
(defun C:MO_RO ( / en1 en2  p1 p2 ang1 ang2 ang3)
❶ (setq  en1 (entsel  "¥n オブジェクトを選択 : "))  ; 図形を選択する          → Step1
❷ (setq  en2 (car  en1))  ; 選択した図形を変数 <en2> に代入する
❸ (setq p1  (getpoint "¥n 移動の基点を指示 :"))  ; 基点をセットする          → Step2
❹ (setq ang1(getangle p1 "¥n 方向を指示 :"))  ; 回転の方向を指示する        → Step3
; 移動先の点と方向を指示して、移動距離と回転角度を取得する
❺ (setq p2 (getpoint "¥n 新しい基点を指示 :"))  ; 移動先の点を取得する      → Step4
❻ (setq ang2(getangle p2 "¥n 方向を指示 :"))  ; 回転角度を取得する          → Step5
; ラジアンを度数に変換する
❼ (setq ang3(/ (* (- ang2 ang1) 180.0) pi))  ;ang1 と ang2 の角度の差       → Step6
❽ (command "MOVE" en2 "" p1 p2)  ; 選択した図形を p1 から p2 へ移動する    → Step7
❾ (command "ROTATE" en2 "" p2 ang3)  ;p2 を基点に四角形を回転する           → Step8
❿ (princ)
 ) ;end
```

番号	関数名	説　明
①	entsel	回転移動する図形を選択します。
②	(car en1)	car 関数で、選択した図形の図形名を取得します。
③	getpoint	回転移動の基点を指示します。
④	getangle p1	回転の方向を指示します。（位置合わせの基準線になります。）
⑤	getpoint	移動先の基点を指示します。③の p1 がこの位置にきます。
⑥	getangle p3	回転する角度を取得します。
⑦	(/ (* (- ang2 ang1) 180.0) pi)	回転角度のラジアンを度数に変換します。⑨の [ROTATE] コマンドは、度数を使うためです。
⑧	MOVE	選択した図形を移動します。（基点③の p1 が目的点⑤の p2 へ）
⑨	ROTATE	移動した図形を回転します。
⑩	princ	最後の行に、<nil> を表示させません。

entsel は、P1-204 を参考

Point!

Polar 関数と ROTATE（回転）コマンドの角度について

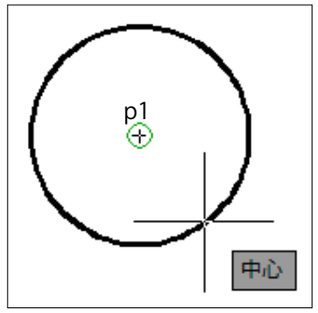
① Polar 関数で使用する角度はラジアンです。(polar p1 ang1 dis1)
ang1 に入る数値はラジアンですから、度数の場合はラジアンに変換します。
(* pi (/ a 180.0)) または、(/ (* pi a) 180.0)

② ROTATE（回転）コマンドで使用する回転角度は度数です。(ROTATE 基点　回転角度)
回転角度に入る数値は度数ですから、ラジアンの場合は度数に変換します。
(* (/ a pi) 180.0) または、 (/ (* 180.0 a) pi)

Getpoint と Lastpoint

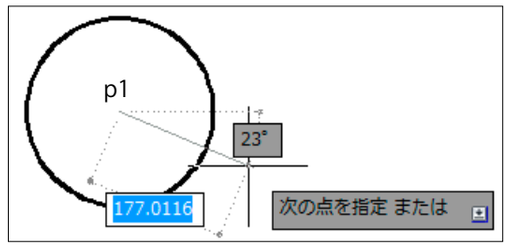
1. コマンドラインから、以下のように入力します。
　コマンド : (setq p1 (getpoint))
この式は、コマンドラインを空白にして、点の入力を待ちます。

2. マウスで円の中心を選択します。
コマンドラインに円の中心座標が表示されます。
_cen どこの (1252.02 998.001 0.0)



3. キーボードから、!p1 と入力すると、点 p1（円の中心）の座標値が表示されます。
コマンド : !p1
(1252.02 998.001 0.0)

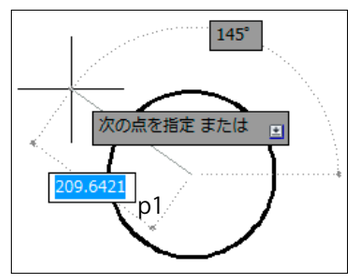
4.[線分] コマンドを選び、コマンドラインに < 線の開始 : > のメッセージが出たときに、
キーボード から、!p1 と入力すると、先ほどの点から線分が開始されます。



5. システム変数 [Lastpoint] は、最後の入力座標値を取得します。
(setq last_pt (getvar "lastpoint")) とすると最後の座標値が、変数 <last_pt> にセットされます。

線分コマンドでキーボードから <!last_pt> と入力すると、上記と同じ結果になります。

キーボードから、<@> を入力すると同じ結果になります。
<@> は「最後の入力点から」という意味です。



AutoCAD と対話する

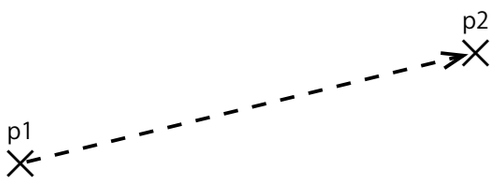
Getpoint 関数と Distance 関数で距離を計算

1 点目と 2 点目を getpoint で指示して、2 点間の距離を求める。


```
(setq p1 (getpoint "¥n1 点目を指示："))          → getpoint で 1 点目の座標を取得
(setq p2 (getpoint p1 "¥n2 点目を指示："))       → getpoint で 2 点目の座標を取得

(setq dist (distance p1 p2))                    → distance で 2 点間の距離を計算
(princ dist)                                    → コマンドラインに結果を表示
```

Getdist 関数で距離を計算（マウス入力）

1 点目を getpoint、2 点目の getdist をマウスで指示して、2 点間の距離を求める。

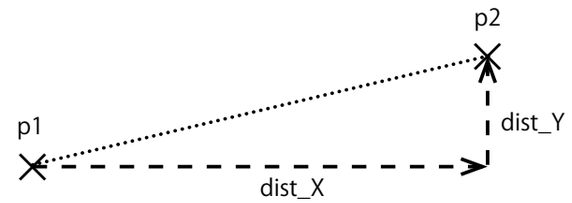



```
(setq p1 (getpoint "¥n 一点目を指示："))          → getpoint で 1 点目の座標を取得
(setq dist (getdist p1 "¥n 二点目を指示："))      → マウスで 2 点目を指示
(princ dist)                                    → コマンドラインに結果を表示
```

getdist の後に P1 をつけると、P1 からラバーバンドが表示されます。

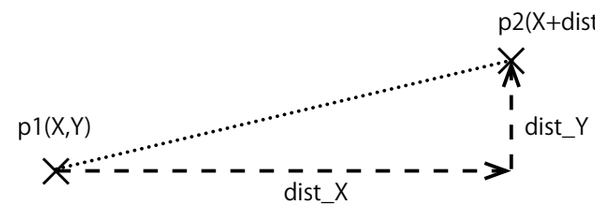
Getdist 関数で距離を計算（キーボード入力）

1 点目を getpoint、2 点目の getdist をキーボードから横と縦の長さを入力して 2 点間の距離を求める。

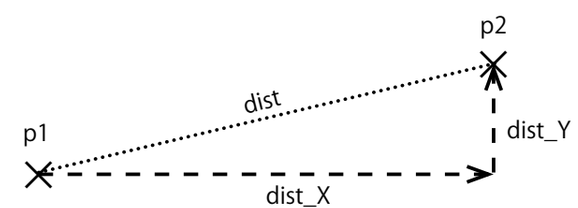



```
(setq p1 (getpoint "¥n1 点目を指示："))          → getpoint で 1 点目の座標を取得
(setq dist_X (getdist "¥n 横の長さ："))          → キーボードから横の距離を入力
(setq dist_Y (getdist "¥n 縦の長さ："))          → キーボードから縦の距離を入力

❶(setq p2 (list (+ (car p1) dist_X) (+ (cadr p1) dist_Y)))   → 横と縦の距離から、p2 の座標を計算
❷(setq dist (sqrt (+ (* dist_X dist_X) (* dist_Y dist_Y))))  → 横と縦の距離から、2 点間の距離を計算
(princ dist)                                                 → コマンドラインに結果を表示
```

① p2 の X 座標は、p1 の X 座標値に dist_X を加えた数値です。
p1 の X 座標を取り出す関数は CAR です。　→ p2 の X 座標は、(+ (car p1) dist_X) です。
同様に、p2 の Y 座標は、p1 の Y 座標値に dist_Y を加えた数値です。
p1 の Y 座標を取り出す関数は CADR です。　→ p2 の Y 座標は、(+ (cadr p1) dist_Y) です。

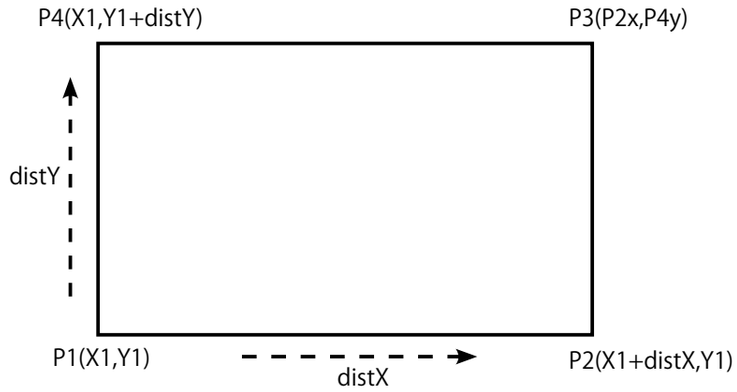


② p1 と p2 の直線距離 (dist) は、ピタゴラスの定理より、$(dist)^2 = (dist\_X)^2 + (dist\_Y)^2$ が成立します。
これを LISP 式で表すと、dist = (sqrt ((* dist_X) (* dist_X) + (* dist_Y) (dist_Y))) となります。
又は dist = (sqrt (+ (expt dist_X 2) (expt dist_Y 2))) です。



Box2.lsp	四角形を作図する
目的：	四角形の横の長さと縦の長さを求めることによって、四角形を作図します。
主要関数：	<getpoint><getdist><list><car><cadr><LINE>

マウスで点 P1(X1,Y1) を指示し、横の長さと縦の長さを入力することによって、他の 3 点 (P2,P3,P4) 求めます。



Step1 ー最初にマウスで点 P1 を指示します。❶

Step2 ーユーザーに横の長さ (distX) を入力させます。❷

Step3 ー同じように、縦の長さ (distY) を入力させます。❸

Step4 ー点 P2 の X 座標は、点 P1 の X 座標に distX を加えたものです。また、Y 座標は、点 P1 の Y 座標と同じです。❹

Step5 ー点 P4 の X 座標は、点 P1 の X 座標と同じです。また、Y 座標は、点 P1 の Y 座標に distY を加えたものです。❺

Step6 ーしたがって、点 P3 の座標は、X の座標が (car P2)、Y の座標が (cadr P4) になります。❻
つまり、(setq P3 (list (car P2) (cadr P4))) になります。

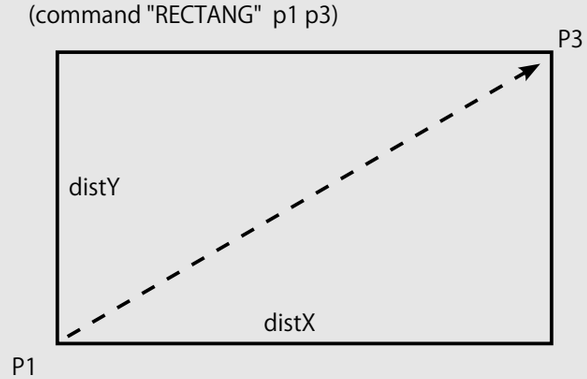
Step7 ー線分コマンドで、p1、p2、p3、p4 と順番に作図します。❼


```
(defun C:Box2( / p1 p2 p3 p4 distX distY)
❶  (setq p1 (getpoint "¥n 一点目を指示：" ))  ; 四角形の左下を指示する          → Step1
❷  (setq distX (getdist p1 "¥n 横の長さ：" ))  ;p1 からの横の長さを入力する     → Step2
❸  (setq distY (getdist p1 "¥n 縦の長さ：" ))  ;p1 からの縦の長さを入力する     → Step3
   ;p1 からの横と縦の長さから、p2、p3、p4 の座標を計算する
❹  (setq p2 (list ( + (car p1) distX) (cadr p1)))  ;p2 の座標を計算する         → Step4
❺  (setq p4 (list  (car p1) ( + (cadr p1) distY)))  ;p4 の座標を計算する        → Step5
❻  (setq p3 (list (car p2) (cadr p4)))  ;p3 の座標を計算する                    → Step6
   ; 線分コマンドで作図する
❼  (command "LINE" p1 p2 p3 p4 "C" )  ; 最後の "C" は Close[ 閉じる ] の C     → Step7
) ;end
```

番号	関数名	説　明
①	getpoint	コマンドラインに表示される文字を見て、ユーザーが指示した点の座標を変数 p1 にセットします。
②	getdist p1	ユーザーがキーボードから距離を入力するか、マウスで 2 点目を指示した点か ら計算して、その長さを変数 distX にセットします。
③	getdist p1	変数 distY も同様に取得します。
④	setq p2	点 p1 の座標と distX から、点 p2 の座標を作ります。
⑤	setq p4	点 p1 の座標と distY から、点 p4 の座標を作ります。
⑥	setq p3	点 p2 と点 p4 の座標から、点 p3 の座標を作ります。
⑦	LINE	4 つの点が取得できたので、線分コマンドで四角形を作図します。 最後の "C" は最初の点 (P1) まで結ぶ (Close) する意味です。

Point!

p1 と p3 だけの座標を使って四角形が作図できます。
長方形 [RECTANG] を使用します。
p3 は (list (+ (car p1) distX) (+ (cadr p1) distY)) より
(command "RECTANG" p1 p3)



3 ユーザー入力関数（キーワード）

Initget	get 関数で使用するオプションを指定します。

get 関数の前の行で指定します。オプションは、ビットコードの合計で指定します。

```
(Initget (+ 1 2))       < ユーザーが <0> 以外の整数を指定するまで、何度でも繰り返します。>
(setq int_No (getint "¥n 整数を入力してください："))                    <¥n は改行コード >
```

ユーザーに整数だけを入力するように制限します。

initget 関数のビットコードを組み合わせて使います。(1 + 2 = 3)<initget 3> でも可。

1 → Null の入力 (改行のみ) を受け付けません。

2 → 0(ゼロ) の入力を受け付けません。

ここで <0> や [Enter< 改行 >] を押すとエラーメッセージが表示されます。

< 整数値を入力してください .>

Initget で入力データを制限する

(initget [ビットコード] [文字列リスト])

Initget のビットコードと意味	
ビットコード	説　明
1	Null の入力 (改行のみ) を受け付けません。
2	0(ゼロ) の入力を受け付けません。
4	負の入力を受け付けません。
8	LIMCHECK が ON であっても図面範囲を無視します。
16	(現在は使用されていない。)
32	ラバーバンドまたはボックスを描くときに破線で表示します。
64	3 次元点の z 座標を無視します。(getdist 関数だけに有効)

☞上記以外では、[128]、[256]、[512]、[1024] があります。

Get 関数と有効なビットコード						
関数	null 入力 禁止 （1）	0(ゼロ) 入力 禁止 （2）	マイナス入力 禁止 （4）	図面範囲 無制限 （8）	破線の使用 （３２）	2 次元の距離 （６４）
Getint	●	●	●			
Getreal	●	●	●			
Getdist	●	●	●		●	●
Getangle	●	●			●	
Getorient	●	●			●	
Getpoint	●			●	●	
Getcorner	●			●	●	
Getkword	●					
Getstring	キーワードは指定できません。					
getvar	キーワードは指定できません。					

Getkword	Initget 関数で指定したキーワードを受け付けます。

```
(Initget 1 "Yes  No")    → 1 は空打ちを受け付けません。"Y" 、"y"、"N"、"n" を受け付けます。
(setq ans (getkword " よろしいですか？  Yes/No："))
  (if (or (= ans "Y") (= ans "y"))                → キーボードからの入力が、"Y" か "y" の時。
      (command "LINE" pt1 pt2 "")                 →T（"Y" か "y"）であれば、線分を作図する。
      (command "CIRCLE"  pt1 rad1)                → nil（"N" か "n"）であれば、円を作図する。
  );if
```

Getkword は、Initget と組み合わせて使います。Initget 関数で指定した値だけが、
Getkword に入力できます。それ以外の値を入力すると、入力のやり直しを要求されます。

キーワードの引数には、次の規則があります。

①キーワードごとに 1 つ以上のスペースが必要です。

例として、("Yes No") や ("Line" "Circle" "Rectang") など。

②キーワードには、半角英数字とハイフン (ー) だけが使用できます。

③ Getkword と Initget の組み合わせは以下のようになります。

```
(Initget 1 "Yes  No")
(setq ans (getkword " よろしいですか？  Yes/No："))
```

次の文字がコマンドラインに表示されます。

よろしいですか？ Yes/No：

④このプロンプトには以下の文字だけが入力できます。

"Y" "y" "Yes" "N" "n" "No"

つまり、Initget で指定したキーワード全部か大文字部分の入力だけが許されます。

キー入力は、大文字でも小文字でも構いません。

⑤キーワードの中の大文字は、複数あっても構いませんが連続していなければなりません。

Yes YEs yES YES は OK ですが、yes YeS は受け付けません。

⑥オペレーターに判りやすくするために次のような工夫も良いでしょう。

(initget "Yes No") → (initget "Y= はい N= いいえ ")

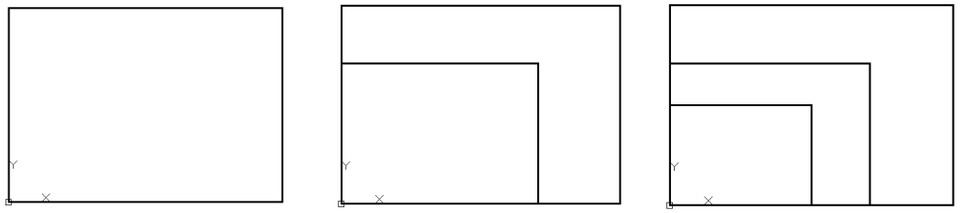
(initget "On Off") → (initget "On= する Off= しない ")

initget 関数と getkword 関数の使用例

ユーザーに <1> か <2> か <3> の入力を促し、ケースに従って A1 サイズから A3 サイズの四角形を作図します。

```
(defun  C:waku_1( / waku)
❶  (initget  1 "1  2  3")
❷  (setq  waku  (getkword  "¥n 用紙の大きさを指定 (1=A1  2= A2  3=A3) : "))
    (cond
❸     ((= waku  "1")  (command  "RECTANG"  "0,0"  "841,594"))   ← A1 サイズの枠を作図
❹     ((= waku  "2")  (command  "RECTANG"  "0,0"  "594,420"))   ← A2 サイズの枠を作図
❺     ((= waku  "3")  (command  "RECTANG"  "0,0"  "420,297"))   ← A3 サイズの枠を作図
    ) ;cond
) ;end
```

キーワード =1　　キーワード =2　　キーワード =3



```
コマンド: WAKU_1
用紙の大きさを指定(1=A1  2= A2  3=A3)：1
RECTANG
一方のコーナーを指定 または [面取り(C)/高度(E)/フィレット(F)/厚さ(T)/幅(W)]: 0,0
もう一方のコーナーを指定 または [面積(A)/サイズ(D)/回転角度(R)]: 841,594
コマンド:
コマンド:
WAKU_1
用紙の大きさを指定(1=A1  2= A2  3=A3)：2
RECTANG
一方のコーナーを指定 または [面取り(C)/高度(E)/フィレット(F)/厚さ(T)/幅(W)]: 0,0
もう一方のコーナーを指定 または [面積(A)/サイズ(D)/回転角度(R)]: 594,420
コマンド:
コマンド:
WAKU_1
用紙の大きさを指定(1=A1  2= A2  3=A3)：3
RECTANG
一方のコーナーを指定 または [面取り(C)/高度(E)/フィレット(F)/厚さ(T)/幅(W)]: 0,0
もう一方のコーナーを指定 または [面積(A)/サイズ(D)/回転角度(R)]: 420,297
コマンド:
ここにコマンドを入力
```

番号	関数名	説　明
①	initget	1,2,3 以外の数字は受け付けません。(空打ちは無効)
②	getkword	1,2,3 のいずれかの数字の入力を促します。
③	"RECTANG" "0,0" "841,594")	A1 サイズ (841,594) の大きさの四角形を作図します。
④	"RECTANG" "0,0" "594,420")	A2 サイズ (594,420) の大きさの四角形を作図します。
⑤	"RECTANG" "0,0" "420,297")	A3 サイズ (420,297) の大きさの四角形を作図します。

cond 関数の <T> と <exit>

前ページでは、initget を使ってユーザーの入力を制限しましたが、下記のように initget を使わない場合を考えてみましょう。

❶ <T> を使用する

```
(defun  C:waku_2( / waku)
❶  (setq  waku  (getint  "¥n 用紙の大きさを指定 (1=A1  2= A2  3=A3) : "))
    (cond
❷     ((= waku  1)  (command  "RECTANG"  "0,0"  "841,594"))   ← A1 サイズの枠を作図
❸     ((= waku  2)  (command  "RECTANG"  "0,0"  "594,420"))   ← A2 サイズの枠を作図
❹     ((= waku  3)  (command  "RECTANG"  "0,0"  "420,297"))   ← A3 サイズの枠を作図
❺     (   T         (prompt  "¥n 終了します。"))              ← 1、2、3 以外の時の処理
    ) ;cond
) ;end
```

このプログラムでは、initget と getkword を使っておりません。
①で（1 か 2 か 3）の入力を促していますが、それ以外の数値が入力されることも考慮する必要があります。
この場合、⑤のように <T> の後に処理を記述しておけばエラーにはなりません。
<T> は真に評価されるので、この処理は必ず実行されます。
上記では、<1><2><3> 以外の数値が入力されたときは、コマンドラインに < 終了します。> のメッセージを表示します。

❷ <exit> を使用する

⑤の行を次のように変更します。

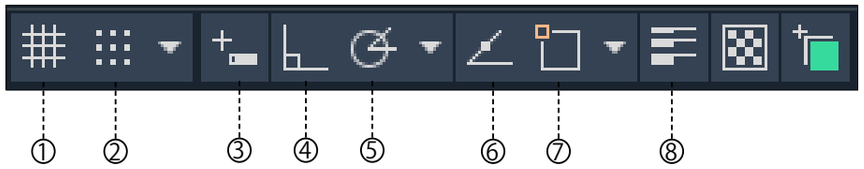
```
(  T   (exit))
```

この場合、AutoCAD は以下のエラーメッセージを表示して、通常のプロンプトに戻ります。

用紙の大きさを指定 (1=A1 2= A2 3=A3：) 5　　→ ユーザーが <5> と入力
; エラー : quit / exit による中止　　→ プログラムは中止されます。

第 3 節　ジオメトリック ユーティリティ

[ジオメトリック ユーティリティ] の 1 つに下図のステータス バーがあります。
このステータス バーのアイコンはクリックするごとに、有効 (On) と無効 (Off) が切り替わります。



① [グリッド表示]・・グリッドの表示 / 非表示を指定します。

0	グリッドを非表示にします。
1	グリッドを表示します。

② [スナップ モード]・・スナップ モードのオン / オフを切り替えます。

0	スナップ モードをオフにします。
1	現在のビューポートでスナップ モードをオンにします。

③ [ダイナミック入力]・・作図ツールチップの外観をコントロールします。

0	ダイナミック入力をオフにします。
1	ダイナミック入力をオンにします。

④ [直交モード]・・カーソルの移動を水平方向および垂直方向に制限します。

0	直交モードをオフにします。
1	直交モードをオンにします。

⑤ [極トラッキング]・・ユーザー指定の極角度で定義した一時的な位置合わせパスを表示します。

0	極トラッキングをオフにします。
1	極トラッキングをオンにします。

⑥ [オブジェクト スナップ トラッキング]・・トラッキング パスの表示をコントロールします。

0	オブジェクト スナップ トラッキングをオフにします。
1	オブジェクト スナップ トラッキングをオンにします。

⑦ [オブジェクト スナップ]・・定常オブジェクト スナップの設定をコントロールします。

0	定常オブジェクト スナップをオフにします。
1	定常オブジェクト スナップをオンにします。

⑧ [線の太さを表示 / 非表示]・・現在の図面で線の太さを表示するかどうかをコントロールします。

0	線の太さは表示しません。
1	線の太さを表示します。

使用頻度の高い AutoLISP のユーザー入力関数を、次の表に示します。（順不同）

①ジオメトリック ユーティリティ関数	
関　数	説　明
OTHOMODE	カーソルの移動を水平方向および垂直方向に制限します。
OSNAP	指定された点に、オブジェクト スナップ モードを適用します。

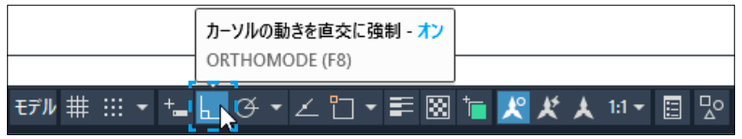
1　ジオメトリック ユーティリティ関数

OTHOMODE	カーソルの移動を水平方向および垂直方向に制限します。

ORTHOMODE がオンの場合、カーソルは UCS および現在のグリッド回転角度に対して水平または垂直にしか移動できません。

(OTHOMODE 0)　→ 直交モードをオフにします。
(OTHOMODE 1)　→ 直交モードをオンにします。

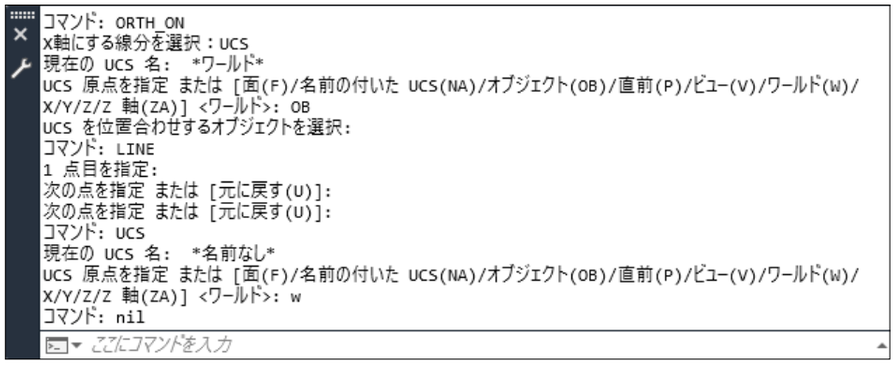
ステータス バーにある直交モード



（例 1）　斜めの線分に平行な線分を作図します。

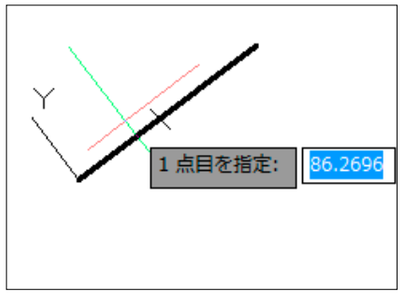
```
(defun  C:orth_on ( / en1)
   (setq en1 (entsel "¥nX 軸にする線分を選択："))   →平行にしたい線分を選択します。
   (setq en1 (car en1))                              →（注 1）図形名を選択します。
   (command "UCS"  "OB"  en1)                        → UCS のオプションで <OB> を選びます。
   (setvar  "ORTHOMODE"  1)                          → "ORTHOMODE" を <1> に切り替えます。
   (command  "LINE" pause pause "")                  →線分コマンドを使います。
   (setvar  "ORTHOMODE"  0)                          → "ORTHOMODE" を <0> に戻します。
   (command "UCS"  "w")                              →ワールド座標系に戻します。
)
```

（注 1）entsel 関数でオブジェクトを選択すると、以下のような情報を取得します。
(< 図形名 : 7ef02968> (56.7275 74.2105 0.0))
これは、図形名と選択した点座標のセットになっています。これから、1 番目のリストを取得する関数 <car> を使って、線分の図形 < 図形名 : 7ef02968> を変数 en1 にセットします。



（例 2） 斜めの線分に垂直な線分を作図します。

Step1

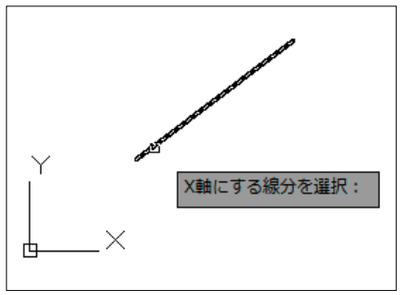


①選択した線分に水平または垂直の線分を作図します。
対象となる線分を選択します。
(setq en1 (entsel "¥nX 軸にする線分を選択："))

② ent1 の情報は、図形名と指示した座標の組み合わせですから、図形名だけを取り出して、変数 en1 にセットします。car 関数で図形名を取得できます。

entsel は、P1-204 を参考

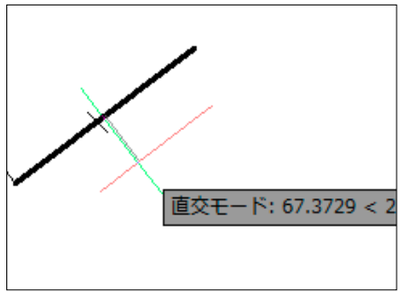
Step2



③ UCS コマンドのオプション（オブジェクト <OB>）を選び、選択した図形（線分）を指定します。

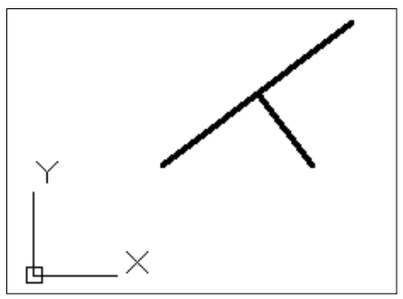
④ (setvar "ORTHOMODE" 1)
選択した線分との垂直線を作成するために、カーソルを直交モードに切り替えます。

Step3



⑤ (command "LINE" pause pause "")
線分コマンドを使い、垂直線を作図します。pause はユーザーがマウスで指示するのを待ちます。
線分コマンドは、1 点目と 2 点目を指示しますから、pause を 2 回使います。

Step4



⑥ (setvar "ORTHOMODE" 0)
"ORTHOMODE" を元の <0> に戻します。

⑦ (command "UCS" "w")
ユーザー座標系も、元のワールド座標系に戻します。

2 オブジェクト スナップ（文字列で指示）

OSNAP	指定された点に、オブジェクト スナップ モードを適用します。

osnap 関数は、AutoCAD のオブジェクト スナップ モードの 1 つを使用して、点を検出します。
オブジェクト スナップ モードは、文字列引数で指定します。

次の osnap 関数呼び出しは、pt1 の近くのオブジェクトの中点を検出します。

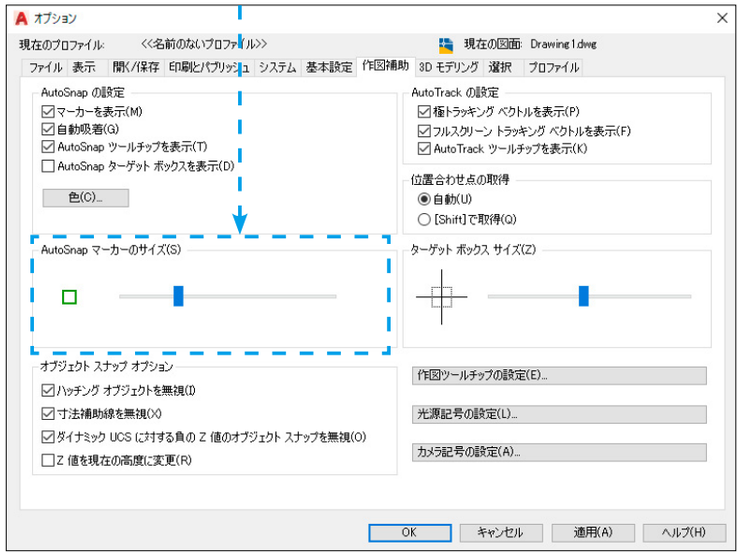
```
(setq pt2 (osnap pt1 "midp"))
```

次の呼び出しは、pt1 に最も近いオブジェクトの中点、端点、または中心点を検出します。

```
(setq pt2 (osnap pt1 "midp,endp,center"))
```

どちらの例でも、osnap の条件に合うものが見つかった場合、pt2 にそのスナップ点が代入されます。複数のスナップ点が条件を満たす場合、点はシステム変数 SORTENTS の設定にしたがって選択されます。条件に合う点が見つからなかった場合、pt2 は nil に設定されます。

オブジェクト スナップを使用する場合、システム変数 APERTURE によって、指定した点からオブジェクトまでの許容可能な近接範囲が決まります。



文字列で指示した場合：(setq pt2 (osnap pt1 "midp"))
→ 一時的なので、解除する必要はありません。

3 オブジェクト スナップ（システム変数で設定）

AutoCAD のオブジェクト スナップ モードをシステム変数 [OSMODE] で指定できます。

OSMODE で指定するには、システム変数に代入する関数 [setvar] を使います。

(setvar "osmode" 1)　　→ <1> は端点をスナップします。

AutoLISP のプログラムでは、最初に現在のスナップ モードを取得しておいて、最後に戻す処理をすることが重要です。

① (setq old_osnap (getvar "osmode"))　　→ 現在のスナップ モードを変数 <old_osnap> に格納します。
・
② (setvar "osmode" 3)　　→ スナップ モードを新規に設定します。
・
・
③ (setvar "osmode" old_osnap)　　→ プログラムの終了前に、スナップ モードを元に戻します。

osmode にセットする数値コード	
ビットコード	モード
0	解除 [NON]（モード設定なし）
1	端点 [END]
2	中点 [MID]
4	中心 [CEN]
8	点 [NOD]
16	四半円点 [QUA]
32	交点 [INT]
64	挿入基点 [INS]
128	垂線 [PER]
256	接線 [TAN]
512	近接点 [NEA]
1024	図心 [GCE]
2048	仮想交点 [APP]
4096	延長 [EXT]
8192	平行 [PAR]
16384	現在の定常オブジェクト スナップを無効

Point!

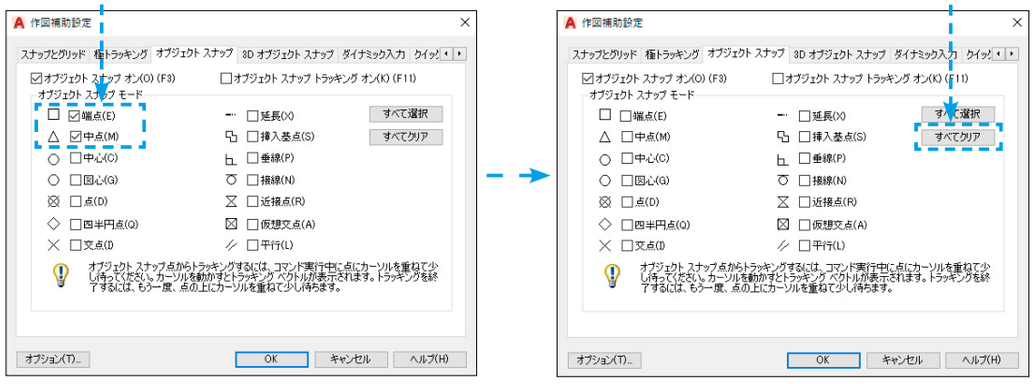
osmode のビットコード <0> は [解除]、<16384> は [無効] ですが、両方とも同じ結果になります。

オブジェクト スナップを複数指定する場合は、これらの値の和を入力します。たとえば、20 と入力すると、[中心](ビットコード 4) と [四半円点](ビットコード 16) のオブジェクト スナップを指定したことになります。また、<15359> と入力すると、すべてのオブジェクト スナップを指定したことになります。

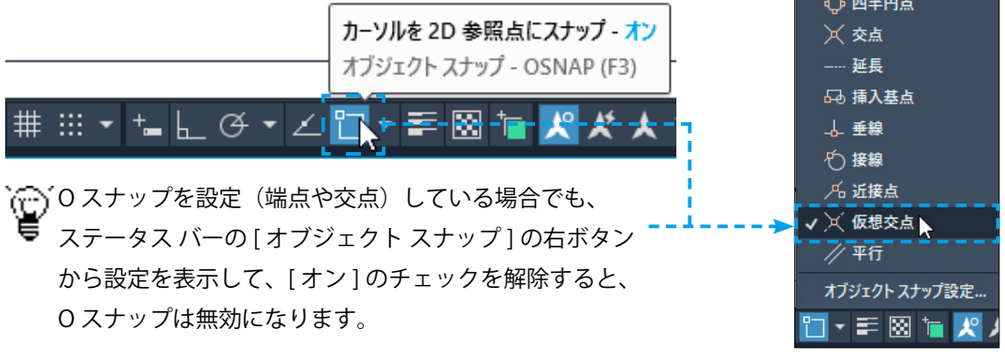
複数のオブジェクト スナップを使う時の組み合わせ例	
ビット コード	モード
3(1+2)	端点 + 中点
5(1+4)	端点 + 中心
7(1+2+4)	端点 + 中点＋中心
9(1+8)	端点 + 点
10(2+8)	中点 + 点

(setvar "osmode" 3) にセットすると、
[作図補助設定] ダイアログの < 端点 > と < 中点 > がチェックされています。

(setvar "osmode" 0) にセットすると、
[作図補助設定] ダイアログの < 端点 > と < 中点 > のチェックが解除されています。



数値で指示した場合：(setvar "osmode" 3)
この後、O スナップモードを解除またはクリアするには、
解除　→ (setvar "osmode" 0)
クリア→ (setvar "osmode" 1024)

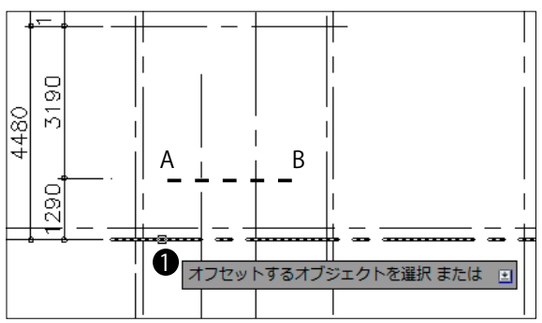


O スナップを設定（端点や交点）している場合でも、ステータス バーの [オブジェクト スナップ] の右ボタンから設定を表示して、[オン] のチェックを解除すると、O スナップは無効になります。

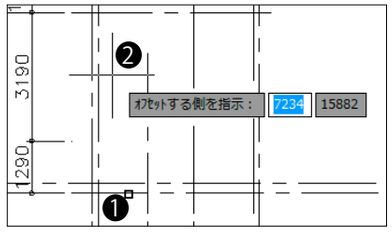
4 総合プログラム I

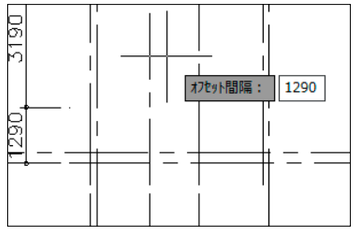
off_3 .lsp	平行線の長さをマウスで指定する
目的：	オフセットする線分を選択し、オフセット距離を入力します。 そのあと、マウスで長さを指定します。
主要関数：	<entsel><getpoint><getdist><car><entget><assoc><trans><angle><polar> <inters><LINE><UCS>

手順 1 ー下図の①の水平線を上側へ 1290 ミリ平行複写し、その長さを A-B にします。

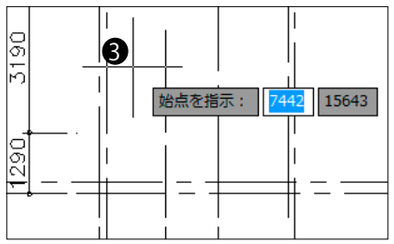


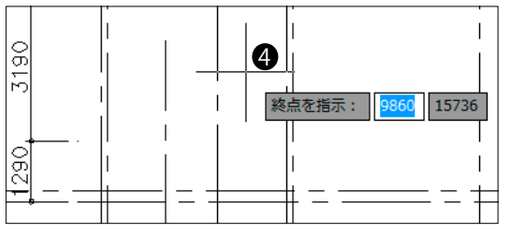
手順 2 ー（左）オフセットする側は上側を指示し②、(右）オフセット間隔を <1290> と入力します。



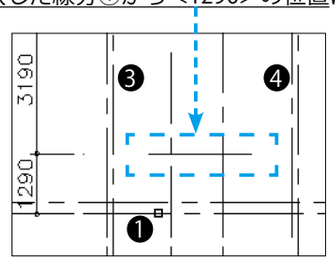


手順 3 ー（左）平行線の始点の位置③を指示します。（右）平行線の終点の位置④を指示します。





手順 4 ー選択した線分①から <1290> の位置に、指定した長さ (③ - ④) だけの線分が作図されます。

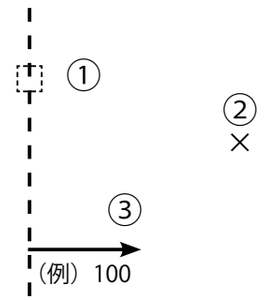


AutoCAD のオフセットコマンドは選択した線分と同じ長さの線分が作成されます。
このコマンドは、マウスで線分の長さを指定することができます。

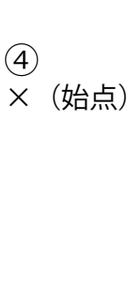
手順図 1・2
①オフセットしたい線分を選択します。

②オフセットの方向をマウスで指示します。

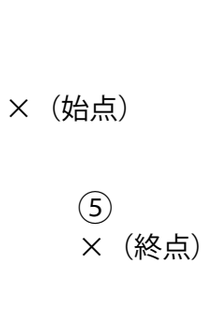
③キーボードからオフセット間隔を入力します。



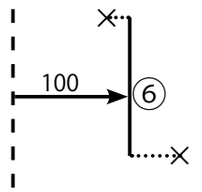
手順図 3（左）
④平行線の始点の位置を指示します。



手順図 3（右）
⑤平行線の終点の位置を指示します。

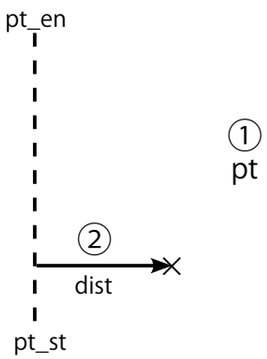


手順 4
⑥最初に指示した距離 (100) のところに、マウスで
指示した長さだけの平行線が作図されます。

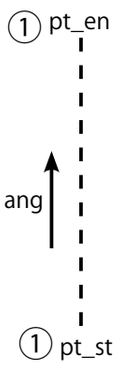


Step1 ーオフセットする基準線を選択し、そのオブジェクトを変数 <ent> にセットします。❶❷
この ent は図形名と座標値の組み合わせです。Step3 で始点と終点の座標を取得します。

Step2 ー① “オフセットする側を指示：” のメッセージを出して、オフセットの方向をマウスで指示し、その座標値を変数 <pt> にセットします。❸
② “オフセット間隔：” のメッセージを出して、オフセット間隔を変数 <dist> にセットします。dist 関数はキーボードからでも、マウスで画面を指示しても入力できます。❹

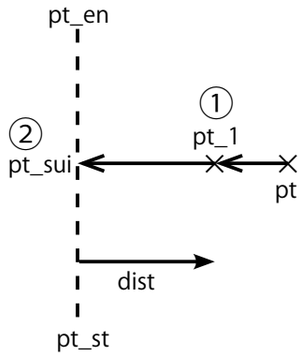


Step3 ー①始点 (pt_st) と終点 (pt_en) の座標を求めます。
ユーザー座標系に変更します。図形を作成したり編集するときは座標系をワールド座標系からユーザー座標系に変更しておきます。
❺❻❼❽
②ユーザー座標系での始点と終点の角度 (ang) を取得します。❾

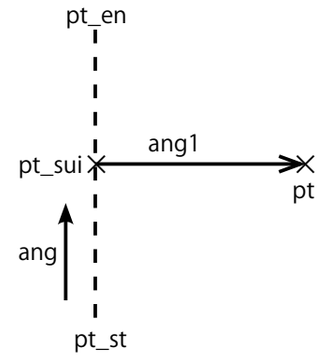


この場合は、必ずしも tarns 関数は必要有りませんが既存の図形を利用して作図する場合は、trans 関数で座標変換しておく方が安全です。

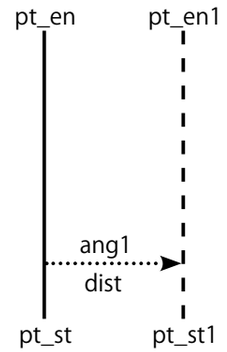
Step4 ー①選択したオブジェクトに直角方向の点 (1.0 は仮の点）を決めます。❿
②選択したオブジェクトと点 pt と点 pt_1 を結ぶ線との交点 (pt_sui) を求めます。⓫



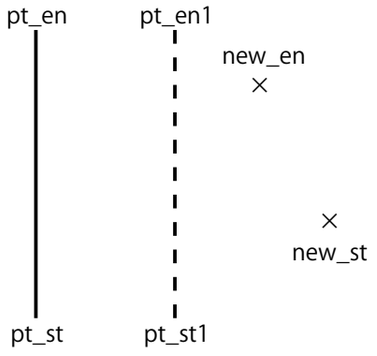
Step5 ー pt_sui から pt の角度 (ang1) を求めます。⓬



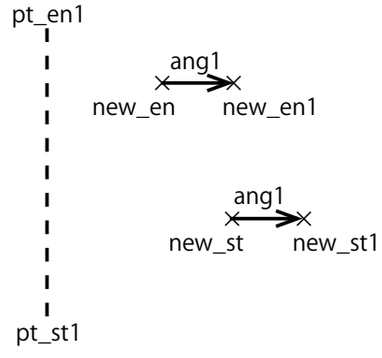
Step6 ー①選択したオブジェクトの始点 (pt_st) から角度 ang1、距離 dist の位置に仮想線分の始点 (pt_st1) の点を求めます。⓭
②同様に、選択したオブジェクトの終点 (pt_en) から角度 ang1、距離 dist の位置に仮想線分の終点 (pt_en1) の点を求めます。⓮



Step7 ー① " 始点を指示：" のメッセージを出して、新しいオフセットの始点を指示します。(new_st) ⓯
② " 終点を指示：" のメッセージを出して、新しいオフセットの終点を指示します。(new_en) ⓰

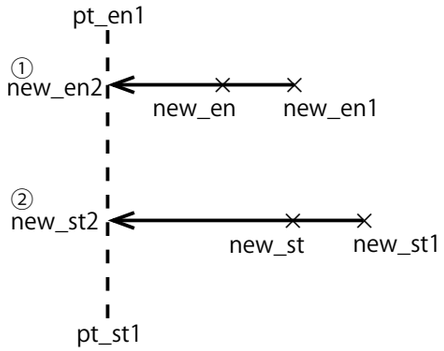


Step8 ー①新しいオフセットの始点 (new_st) から、ang1 の方向に短い線分 (1 ミリ) の位置を new_st1 とします。⓱
②同様に新しいオフセットの終点 (new_en) から、ang1 の方向に短い線分 (1 ミリ) の位置を new_en1 とします。⓲



Step9 ー①選択した線分から dist の距離にある仮想線分 (pt_st1 ー pt_en1) と (new_st ー new_st1) の交点を new_st2 とします。⑲

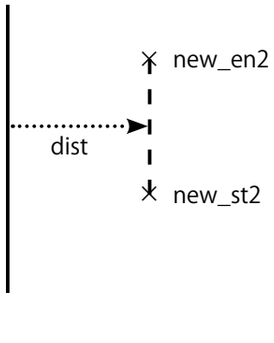
②同様に仮想線分 (pt_st1 ー pt_en1) と (new_en ー new_en1) の交点を new_en2 とします。⑳



Step10 ー① new_st2 と new_en2 を始点、終点として線分を作成します。㉑

② UCS をユーザー座標系からワールド座標系に戻します。㉒

☞UCS は元の座標系に戻しておきます。




```
(defun C:off_3 ( / ent pt dist pt_st pt_en ang pt_1 pt_sui ang1
                   pt_2 pt_3 new_st new_en new_st1 new_en1
                   new_st2 new_en2)
; 基準線を選択し、オフセットする側と間隔を入力します。
❶ (prompt "¥n 基準線を指示してください。") ; メッセージを表示する
❷ (setq ent (entsel)) ; オフセットする線分を選択する
❸ (setq pt (getpoint "¥n オフセットする側を指示：")) ; オフセット側を指示
❹ (setq dist (getdist "¥n オフセット間隔：")) ; オフセット間隔を指定する
; 基準線の角度と pt への角度を計算します。
❺ (setq pt_st (cdr (assoc 10 (entget (car ent))))) ; 始点の座標を取得する
❻ (setq pt_en (cdr (assoc 11 (entget (car ent))))) ; 終点の座標を取得する
❼ (setq pt_st (trans pt_st 0 1)) ; 始点の座標を UCS 変換する
❽ (setq pt_en (trans pt_en 0 1)) ; 終点の座標を UCS 変換する
❾ (setq ang (angle pt_st pt_en)) ; 始点から終点への角度を求める
❿ (setq pt_1(polar pt (+ ang (* 0.5 pi)) 1.0)) ; 角度が直角で、1 ミリの位置を計算
⓫ (setq pt_sui (inters pt_st pt_en pt pt_1 nil)) ; 基準線との交点を求める
⓬ (setq ang1 (angle pt_sui pt)) ; 交点からの角度を計算する
```

→ Step1 (❶❷)
→ Step2 (❸❹)
→ Step3 (❺❻❼❽❾)
→ Step4 (❿⓫)
→ Step5 (⓬)


```
⓭ (setq pt_st1 (polar pt_st ang1 dist)) ;pt_st を基点に pt_st1 の座標を計算
⓮ (setq pt_en1 (polar pt_en ang1 dist)) ;pt_en を基点に pt_en1 の座標を計算
; マウスで長さを指定します。
⓯ (setq new_st (getpoint "¥n 始点を指示：")) ; マウスで始点を指定
⓰ (setq new_en (getpoint new_st "¥n 終点を指示：")) ; マウスで終点を指定
; 新しい始点と終点の座標を計算します。
⓱ (setq new_st1(polar new_st ang1 1.0)) ;new_st から 1 ミリの位置が new_st1
⓲ (setq new_en1(polar new_en ang1 1.0)) ;new_en から 1 ミリの位置が new_en1
⓳ (setq new_st2(inters pt_st1 pt_en1 new_st new_st1 nil)) ; 交点を求める
⓴ (setq new_en2(inters pt_st1 pt_en1 new_en new_en1 nil)) ; 交点を求める
; 線分を作成します。
㉑ (command "LINE" new_st2 new_en2 "") ; 指示した長さの線分を作図する
㉒ (command "UCS" "W") ; ワールド座標系に戻す
(princ)
);end
```

→ Step6 (⓭⓮)
→ Step7 (⓯⓰)
→ Step8 (⓱⓲)
→ Step9 (⓳⓴)
→ Step10 (㉑㉒)

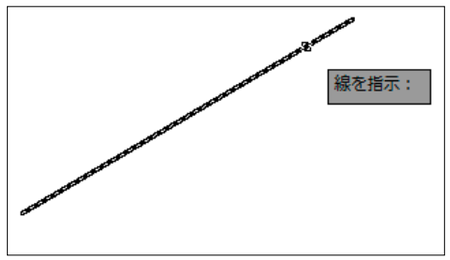
番号	関数名	説　明
①	prompt	“ から “の文字をコマンドラインに表示します。
②	entsel	ユーザーに図形を選択させます。その図形を変数 ent にセットします。
③	getpoint	オフセットする側の点を指定させます。その座標を変数 pt にセットします。
④	getdist	オフセット間隔を入力させます。（キーボード又は図面上からでも構いません。）
⑤	assoc 10	選択した線分の始点の座標を取得し、変数 pt_st にセットします。
⑥	assoc 11	選択した線分の終点の座標を取得し、変数 pt_en にセットします。
⑦	trans	trans 関数は座標変換の関数です。 <trans 0 1> はユーザー座標系に変換します。この後の座標計算はユーザー座標系で行います。
⑧	trans	trans 関数は座標変換の関数です。 <trans 0 1> はユーザー座標系に変換します。この後の座標計算はユーザー座標系で行います。
⑨	angle	選択した線分の角度を求めます。（始点→終点の角度）
⑩	polar	基点 (pt) から角度と長さで、新しい座標を計算します。オフセットは選択した線分の直角方向にありますから、角度は線分の角度に 90 度 (0.5 pi) を加えたものになります。
⑪	inters	2 本の線分の交点座標を求めます。最後に nil を付けると、線分が交わっていなくても仮想の交点座標が求まります。
⑫	angle	選択した線分の直角の角度を計算します。
⑬	polar	pt_st から、角度 ang1、距離 dist の距離の位置を計算します。
⑭	polar	pt_en から、角度 ang1、距離 dist の距離の位置を計算します。
⑮	getpoint	ユーザーに平行線の始点の位置を指示させます。
⑯	getpoint	ユーザーに平行線の終点の位置を指示させます。
⑰	polar	線分 pt_st1,pt_en1 との交点を求めるために、new_st から直角の位置を求めます。
⑱	polar	線分 pt_st1,pt_en1 の交点を求めるために、new_en から直角の位置を求めます。
⑲	inters	新しい始点 (new_st2) の座標を計算します。
⑳	inters	新しい終点 (new_en2) の座標を計算します。
㉑	LINE	新しい始点 (new_st2) と終点 (new_en2) を線分で結びます。
㉒	UCS "W"	最後は元のワールド座標系に戻します。

AutoCAD と対話する

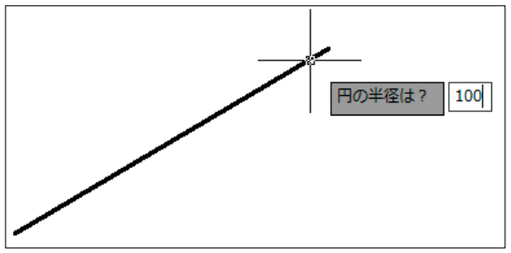
4 総合プログラムⅡ

en_hasi.lsp	線分の端点に接する円を作図する
目的：	選択した線分の端に 2 つの接する円を作図し、選択した円を残します。
主要関数：	<entsel><getreal><cadr><car><entget><assoc><trans><distance><angle> <polar><entlast><equal><redraw><CIRCLE><UCS>

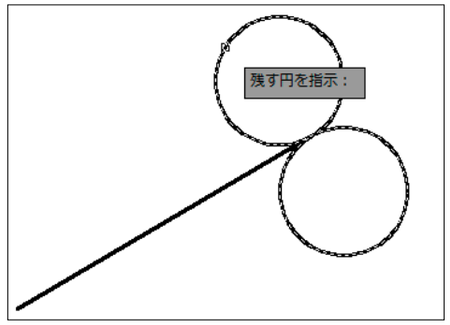
手順 1
接円を作図する線分を選択します。



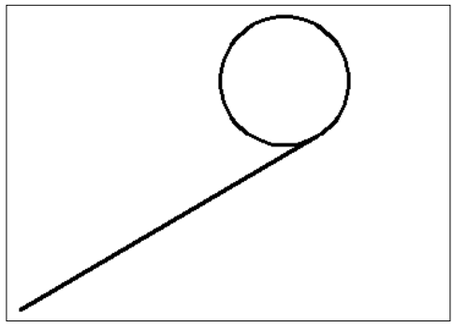
手順 2
円の半径を入力します。



手順 3
線分を指示した点に近い端点に 2 つの接円が作成され、ハイライト表示されます。



手順 4
< 残す円を指示 > のメッセージが出ます。
選択した円が残り、他の円は削除されます。



Step1 ー接円を作図する線分を選択し、線分を指示した座標値を取得します。❶❷
① (entsel " 線を指示:") のメッセージを表示して線分を選択し、変数 <obj_0> に代入します。
②選択したオブジェクトは図形名と座標値のドットペアです。その後半の座標値を変数 <obj_pt> にセットします。Step5 で obj_pt の座標が線分のどちらの端点に近いかを比べるためです。（obj_pt に近い方の端に接円を作図するため）

Step2 ー (getreal " 円の半径は？ ") のメッセージを表示して半径の数値を入力し、変数 <hankei> に代入します。❸

Step3 ー 線分の始点の座標を取得し、ユーザー座標系に変更します。❹❺
(entget (car obj_0) により、線分のドットペアの組み合わせが表示されます。
((-1 . < 図形名 : 7ef03990>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A2") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 971.851 1667.17 0.0) (11 1600.03 1919.45 0.0) (210 0.0 0.0 1.0))
線分の始点の座標は、assoc 関数で (10 971.851 1667.17 0.0) を取り出し、cdr 関数で 2 番目の座標を取り出します。(pt_st に代入します。）
次に trans 関数でユーザー座標系に変更します。

この場合は、必ずしも trans 関数は必要有りませんが
既存の図形を利用して作図する場合は、trans 関数で
座標変換しておく方が安全です。

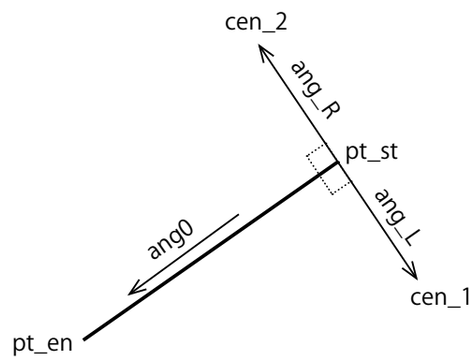
Step4 ー 始点と同様に終点の座標を取得し、ユーザー座標系に変更します。❻❼

Step5 ー 図形を選択した座標値が、この線分の始点座標と終点座標のどちらに近いかを判断します。
近い方を始点に変更します。（始点に近い方に円を作図するため）
もし、（指示した点 <obj_pt> と始点 <pt_st> との距離）が（指示した点と終点 <pt_en> との距離）より大きければ、始点 <pt_st> と終点 <pt_en> を入れ替えます。❽

Step6 ー 改めて、始点から終点への角度を計算し、変数 <ang0> に代入します。❾

Step7 ー 始点の直角方向に円の中心がありますから、その角度をラジアンで求めます。❿⓫
ang0 に 90° 加算した角度を amg_L、90° 減算した角度を ang_R とします。

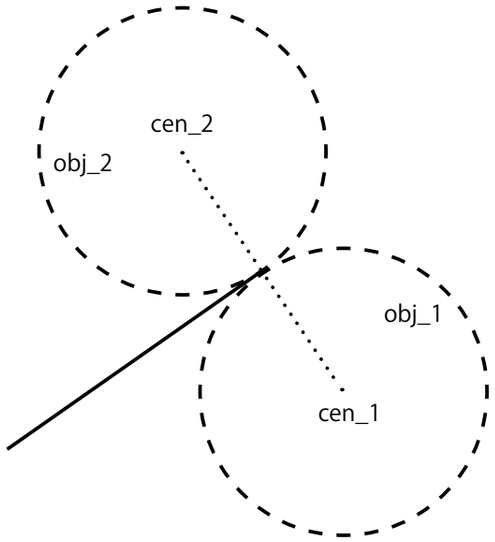
Step8 ー その方向に向かって円の半径の距離が作図する中心になります。⓬⓭



Step9 ー 中心 <cen_1> で円を作図して、変数 <obj_1> にセットします。⑭⑮
entlast 関数は、最後に作図したオブジェクトを取得します。

Step10 ー中心 <cen_2> で円を作図して、変数 <obj_2> にセットします。⑯⑰
entlast 関数は、最後に作図したオブジェクトを取得します。

Step11 ー 2 つの円をハイライト表示させます。⑱⑲
(redraw オブジェクト 3) はオブジェクトをハイライト表示させます。



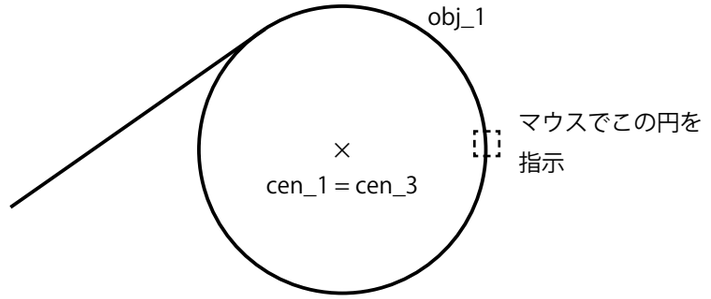
Step12 ー残す円を選択し、変数 <obj_3> に代入します。⑳
(entsel " 残す円を指示：")

Step13 ー 選択した円の中心座標を取得し、変数 <cen_3> にセットします。㉑
(cdr (assoc 10 (entget (car obj_3))))

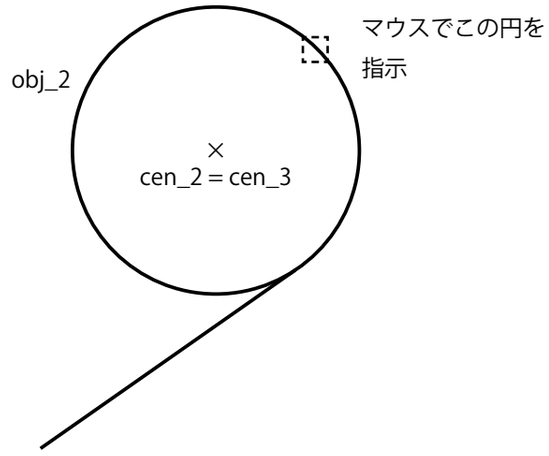
Step14 ー円の中心座標をユーザー座標系に変更します。㉒
(trans cen_3 0 1)

Step15 ー選択した円の中心座標が、ハイライトされている円の中心座標と同じかどうかを比較します。㉓

Step16 ー選択した円の中心座標が第 1 の円の中心座標と同じであれば、第 1 の円のハイライト表示を解除し、第 2 の円を削除します。㉔㉕



Step17 ー選択した円の中心座標が第 1 の円の中心座標と違っていれば、第 1 の円を削除し、第 2 の円のハイライト表示を解除します。㉖㉗



Step18 ーユーザー座標系を初期値のワールド座標系に戻します。㉘

Point!

このプログラムでは UCS を変更していないので、trans 関数を使う必要はありませんが、既存の図形を使って作図をしますので、念のために UCS 座標系に変換して作図を行っています。
Step18 では、ワールド座標系に戻しましたが、直前の座標系に戻す方法もあります。
→ (command "UCS" "P")

AutoCAD と対話する

```
;; 度 ---> ラジアン（Step7 で dtr 関数を使います。）
(defun dtr (a)
   (* pi (/ a 180.0)))
;; ラジアン ---> 度
(defun RTD(a)
   (* 180.0(/ a PI)))

(defun C:en_hasi( / obj_0 hankei obj_pt pt_st pt_en s ang0
                    ang_l ang_r  cen_1 cen_2 obj_1 obj_2 obj_3 cen_3)
; 接円を作図する条件を取得する
❶  (setq obj_0 (entsel "¥n 線を指示："))  ; 接円を作図する線分を選択          → Step1
❷  (setq obj_pt (cadr obj_0))  ; 選択した座標を取得（近い方に円を作図するため）
❸  (setq hankei (getreal "¥n 円の半径は？ "))  ; 円の半径を入力              → Step2
; 選択した線分の 2 つの端点の座標を取得し、obj_pt に近い方を始点にセット
❹  (setq pt_st (cdr (assoc 10 (entget (car obj_0)))))  ;1 つの端点を取得    → Step3
❺  (setq pt_st (trans pt_st 0 1))  ; 念のために、UCS を変換
❻  (setq pt_en (cdr (assoc 11 (entget (car obj_0)))))  ; 残りの端点を取得   → Step4
❼  (setq pt_en (trans pt_en 0 1))  ; 念のために、UCS を変換
❽  (if (> (distance pt_st obj_pt) (distance pt_en obj_pt))                   → Step5
       (setq s pt_st pt_st pt_en pt_en s) ;obj_pt に近い方を
    ) ;if                                  ; 始点 (pt_st) にする
; 線分に直角方向の座標を計算する
❾  (setq ang0 (angle pt_st pt_en))  ; 始点から終点への角度を取得             → Step6
❿  (setq ang_L (+ ang0 (dtr 90)))  ;ang0 から <90°  > の角度を計算          → Step7
⓫  (setq ang_R ( - ang0 (dtr 90)))  ;ang0 から <-90°  > の角度を計算
⓬  (setq cen_1(polar pt_st ang_L hankei))  ;1 つ目の円の中心を計算           → Step8
⓭  (setq cen_2(polar pt_st ang_R hankei))  ;2 つ目の円の中心を計算
; 円を 2 つ作図し、ユーザーが選択した円を残す
⓮  (command "CIRCLE" cen_1 hankei)  ;1 つ目の円を作図                       → Step9
⓯  (setq obj_1 (entlast))  作図した円を取得
⓰  (command "CIRCLE" cen_2 hankei)  ;2 つ目の円を作図                       → Step10
⓱  (setq obj_2 (entlast))  作図した円を取得
⓲  (redraw obj_1  3)  ;1 つ目の円をハイライト表示                           → Step11
⓳  (redraw obj_2  3)  ;2 つ目の円をハイライト表示
```



```
⓴  (setq obj_3 (entsel "¥n 残す円を指示："))  ; 残す円を選択                 → Step12
㉑  (setq cen_3 (cdr (assoc 10 (entget (car obj_3)))))  ; 選択した円の中心   → Step13
㉒  (setq cen_3 (trans cen_3 0 1))  ; 他の円と同様に UCS を変更               → Step14
㉓  (if (equal cen_1 cen_3)  ; 選択した円の中心が、どちらの円であるかを比較   → Step15
        (progn ;1 番目の円と等しければ、以下の処理
㉔          (redraw obj_1  4)  ;1 番目の円のハイライトを解除（円を残す）
㉕          (redraw obj_2  2)  ;2 番目の円を削除                             → Step16
        ) ;progn
        (progn ;2 番目の円と等しければ、以下の処理
㉖          (redraw obj_1  2)  ;2 番目の円のハイライトを解除（円を残す）
㉗          (redraw obj_2  4)  ;1 番目の円を削除                             → Step17
        );progn
    );if
㉘  ("UCS" "W")  ; ユーザー座標系を初期値のワールド座標系に戻す               → Step18
):end
```

番号	関数名	説　明
①	entsel	接円を作図する線分を選択し、変数 [obj_0] にセットします。
②	cadr	線分を選択した座標を変数 [obj_pt] にセットします。
③	getreal	接円の半径を取得します。
④	cdr (assoc 10	選択した線分の端点の座標を取得します。
⑤	trans	UCS でワールド座標系からユーザー座標系に変換します。
⑥	cdr (assoc 11	選択した線分の他方の端点の座標を取得します。
⑦	trans	UCS でワールド座標系からユーザー座標系に変換します。
⑧	(if (>(distance	円を線分の始点から作図するために、近い方を始点座標に切り替えます。
	setq s pt_st	仮の変数 [s] を用意して、始点座標と終点座標の入れ替えを行います。
⑨	angle	始点から終点への角度を計算します。
⑩	(dtr 90)	1 つ目の円の中心座標への角度を計算します。
⑪	(dtr 90)	2 つ目の円の中心座標への角度を計算します。
⑫	polar	1 つ目の円の中心座標を計算します。
⑬	polar	2 つ目の円の中心座標を計算します。
⑭	"CIRCLE"	1 つ目の円を作図します。
⑮	(entlast)	作図した円を取得します。（後でハイライト表示します。）
⑯	"CIRCLE"	2 つ目の円を作図します。
⑰	(entlast)	作図した円を取得します。（後でハイライト表示します。）
⑱	redraw	1 つ目の円をハイライト表示します。
⑲	redraw	2 つ目の円をハイライト表示します。
⑳	entsel	残す円を選択します。
㉑	cdr (assoc 10	選択した円の中心座標を取得します。
㉒	trans	この座標もユーザー座標系に変換し、座標系を揃えます。
㉓	(if (equal	選択した円の中心座標がどちらの円の中心座標と等しいかを比較べます。
㉔	redraw	もし、1 番目の円の中心座標と等しければ、1 番目の円のハイライトを解除。
㉕	redraw	2 番目の円を削除します。
㉖	redraw	もし、2 番目の円の中心座標と等しければ、2 番目の円のハイライトを解除。
㉗	redraw	1 番目の円を削除します。
㉘	"UCS" "W"	ユーザー座標系を初期値のワールド座標系に戻します。

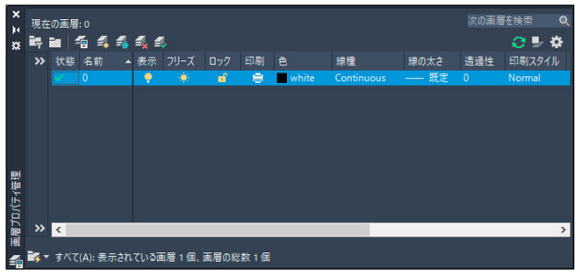
AutoCAD と対話する

第 4 節　テキスト ファイルの入出力

テキスト ファイルを入出力する関数を使うと、図面にある情報をテキスト ファイルにして外部に保存したり、外部のテキスト ファイルの情報を現在の図面に読み込むことができます。

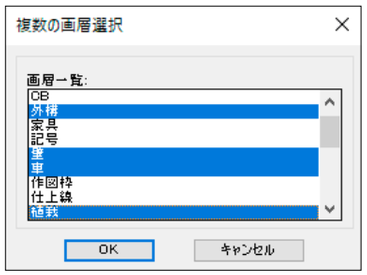
下図は外部の画層ファイルから画層一覧を読み込み、必要な画層名だけを現在開いている図面の画層に追加するものです。第 2 部 3 章 14 節を参考

Step1 ー現在の画層は <0> だけです。

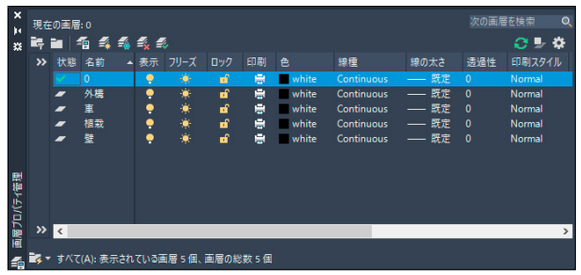


Step2 ー外部のテキストファイルから画層名の一覧を読み込みます。

Step3 ー必要な画層名を選択します。



Step4 ー選択した画層名が取り込まれました。



この節では、このように画層一覧やシステム変数一覧を外部にテキストファイルとして保存したり、呼び出して利用する関数を学びます。

使用頻度の高い AutoLISP のファイル処理関数を、次の表に示します。（順不同）

①ファイル処理関数	
関　数	説　明
(close file-desc)	開いているファイルを閉じます。
(findfile filename)	指定されたファイルを AutoCAD のライブラリ パスで検索します。
(open filename mode)	AutoLISP の入出力関数がアクセスできるようにファイルを開きます。
(read-char [file-desc])	キーボードから 1 文字を読み込みます。ファイル記述子があれば、開いているファイルから 1 文字を読み込みます。
(read-line [file-desc])	キーボードまたは開いているファイルから文字列を 1 行読み込みます。
(write-char num [file-desc])	コマンドラインへ文字コード（整数）で指定する 1 文字を書き出します。ファイル記述子があれば、開いているファイルに書き出します。
(write-line string [file-desc])	コマンドラインに文字列を書き出します。ファイル記述子があれば、開いているファイルに書き出します。

1 主なファイル処理関数

CLOSE	ファイルを閉じて nil を返します

(setq file1 (open "c:¥¥MyLISP.txt" "r") → 開いたファイル (MyLISP.txt) を変数 <file1> にセット。
(close file1) → 開いたファイルを閉じる。

FINDFILE	指定したファイル名を検索します

(findfile "acad.lsp")
（ある場合）→ "C:¥Program Files¥Autodesk¥AutoCAD 2013¥support¥acad.lsp"
（無い場合）→ nil

OPEN	I/O 関数がアクセスできるようにファイルを開きます

(open "FileName" "r") → ファイルの読み込み
(open "FileName" "w") → ファイルの書き込み
(open "FileName" "a") → ファイルへ付加（<a> は小文字）

READ-CHAR	指定したファイルから、1 文字を読み込みます

(read-char file1) → 開いたファイルから 1 文字を読み込む。
(read-char) →キーボードから入力された最初の文字の ASCII 文字を返す。

READ-LINE	指定したファイルから、文字列を読み込みます

(setq file1 (open "c:¥¥MyLISP.txt" "r") → 開いたファイル (MyLISP.txt) を変数 <file1> にセット。
(read-line file1) → 開いたファイルから文字列を読み込む。

WRITE-CHAR	指定したファイルに、1 文字を書き込みます

(write-char ASCII コード file1) → 開いたファイルに 1 文字を書き込む。
(write-char 67) → <C67> がコマンド ウインドウに表示される。

WRITE-LINE	指定したファイルに、文字列を書き込みます

(setq file1 (open "c:¥¥MyLISP.txt" "w") → 開いたファイル (MyLISP.txt) を変数 <file1> にセット。
(write-line " 文字列 " file1) → 開いたファイルに文字列を書き込む。

<¥> 記号はエスケープ文字として使用されるので、文字列内で <¥> 記号を作成するには、2 つの円記号 <¥¥> が必要です。

out_var.lsp	複数のシステム変数の数値を外部ファイルに書き出す
目的：	システム変数を外部ファイル（テキスト ファイル）に書き出します。
主要関数：	<getvar><itoa><strcat><open><write_line><close>

Step1 ー getvar 関数でシステム変数を取得し、変数 <var_osmode> に保存します。❶

(setq var_osmode (getvar "osmode"))

変数 <var_osmode> は数値情報ですが、外部に書き出すには文字列ですので、
この数値を文字列に変換します。
(setq var_osmode (itoa var_osmode))

Step2 ー次にシステム変数名と文字列を連結します。文字列を連結する関数は、strcat です。❷

連結した文字列を保存する変数を <ren_osmode> とします。
(setq ren_osmode (strcat "osmode" var_osmode))

外部ファイルには、例として <osmode3> のように連結されて保存されます。

Step3 ーシステム変数 <LUPREC> も同様に処理します。❸❹

Step4 ーこれらの文字列を順番に外部のテキスト ファイルに書き出します。❺

ファイル名を <out_sys.txt> とします。
(setq f_var (open "out_sys.txt" "w")) → ドライブ名を指定しないときは、AutoCAD 既定の図面フォルダに保存されます。

Step5 ー 外部ファイル < out_sys.txt> に、1 行ずつ順番に書き込みます。❻❼

(write-line ren_osmode f_var) → (write-line 書き込む文字列 ファイル)

元のテキストファイル

変更されたテキストファイル

Step6 ーファイルを閉じます。（必須です）❽

(close f_var)


```
(defun C:out_var ( / var_osmode ren_osmode var_LUPREC ren_LUPREC f_var)
❶  (setq  var_osmode (itoa (getvar  "osmode")))  ; 整数を文字に変換する        → Step1
❷  (setq ren_osmode (strcat  "osmode"  var_osmode))  ;2 つの文字を連結する     → Step2
❸  (setq  var_LUPREC (itoa (getvar  "LUPREC")))  ; 整数を文字に変換
❹  (setq ren_LUPREC (strcat  "LUPREC"  var_LUPREC))  ; 文字を連結              → Step3
Point!
; ファイルを書き込みモードで開く
❺  (setq  f_var  (open  "out_sys.txt"  "w"))  ; 書き込みモードでテキスト ファイルを開く → Step4
;1 行ずつ書き込む
❻  (write-line  ren_osmode  f_var)  ;[ren_osmode] の値を書き込む
❼  (write-line  ren_LUPREC  f_var)  ;[ren_luprec] の値を書き込む              → Step5
; ファイルを閉じる（必須）
❽  (close  f_var)  ; テキスト ファイルを閉じる                                → Step6
 ) ;end
```

番号	関数名	説　明
①	itoa	文字列として外部のファイルに書き出すため、システム変数 "osmode" の数値を文字列に変換します。
②	strcat	"osmode" の文字と文字化された数値を連結します。
③	itoa	文字列として外部のファイルに書き出すため、システム変数 "luprec" の数値を文字列に変換します。
④	strcat	"luprec" の文字と文字化された数値を連結します。
⑤	open "w"	ファイル変数 <f_var> を書き込みモードでオープンします。
⑥	write-line	[ren_osmode] の値を書き込みます。
⑦	write-line	[ren_luprec] の値を書き込みます。
⑧	close	テキスト ファイルを閉じます。これを記述しなければ、書き込みされません。

Point!

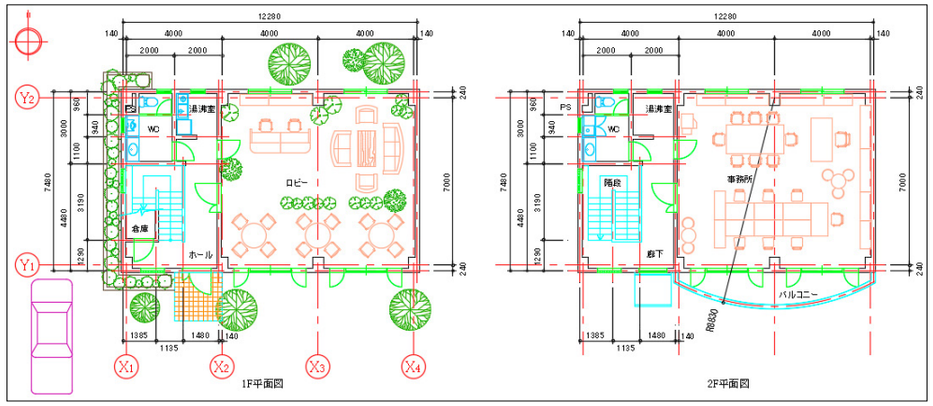
読み込む（書き込む）ファイルをドライブ指定する場合

❺ ファイルの絶対パスを指定しない場合は、AutoCAD の既定の図面フォルダを指定したことになります。（初期値は、ドキュメント）
パスを指定するには、以下のように <¥> を 2 つ続けるか <¥¥>、</> 記号を使います。
例 1：(setq in_layer (open "C:¥¥autolisp¥¥out_layer.txt" "r"))
例 2：(setq in_layer (open "C:/autolisp/out_layer.txt" "r"))

ファイルのディレクトリのパス指定では、スラッシュ記号（/）と円記号（¥）がありますが、¥ を使うときは <C:¥¥myLisp¥¥myText.txt> のように ¥ を 2 つ並べて記述します。

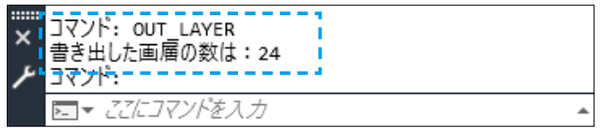
out_layer.lsp	画層名一覧を外部ファイルに書き出す
目的：	システム変数を外部ファイル（テキスト ファイル）に書き出します。
主要関数：	<tblnext><while><assoc><cdr><list><append><nth><open><write_line><1+><close>

Step1 ー画層名の一覧を書き出したい図面を開きます。



Step2 ー [画層プロパティ管理] で存在する画層を確認します。

Step3 ー（左図）out_layer.lsp を実行すると、画層名の一覧が out_layer.txt の名で書き出されます。
（右図）コマンドラインに書き出した画層の数を表示します。




```
(defun C:out_layer( / layname laydata lst new_layer f_layer i)
; 図面にある全画層名を取得する
❶ (setq layname (tblnext "LAYER" T))  ; 最初の画層を取得する
❷ (while layname  ; layname が nil を返すまで繰り返す
❸ (setq laydata (cdr (assoc 2 layname)))  ; layname の画層名を取り出す
❹ (setq lst (append lst (list laydata)))  ; 取得した画層名をリストにする
❺ (setq layname (tblnext "LAYER"))  ; 次の画層を取得する ( 無ければ nil を返す )
  )
; リストにした画層名を順番に外部に書き出す
❻ (setq new_layer (nth 0 lst))  ; リストの 1 番目を取り出す
❼ (setq f_layer (open "out_layer.txt" "w"))  ; 書き込み用のテキスト ファイルを開く
❽ (setq i 1)  ; 書き出した画層数をコマンドラインに表示するためのカウントをセット
❾ (while (/= new_layer nil)  ; 画層名が無くなるまで書き出しを続ける
❿ (write-line new_layer f_layer)  ; 取り出した画層名をテキスト ファイルに書き出す
⓫ (setq new_layer (nth i lst))  ; 次の行の画層名をセットする
⓬ (setq i (1+ i))  ; 書き出した画層の数を <1> 増やして⑨から繰り返す
  );while
⓭ (close f_layer)  ; ファイルを閉じる（必須）
⓮ (prompt "¥n 書き出した画層の数は：")  ; コマンドラインにメッセージを表示する
  (princ ( - i 1))(princ)  ; 画層の数を表示する
);end
```

番号	関数名	説　明
①	tblnext "LAYER" T	画層テーブルにある最初の画層を取得します。（最初の画層名は <0>）
②	while layname	2 番目以降の画層名がある限り、以下のコードを繰り返します。
③	cdr (assoc 2 layname)	画層のドット・ペア <2> から後半部分 < 画層名 > を取り出して変数 edata にセットします。
④	append lst (list laydata)	取り出した画層名を順番にリストにしていきます。 append は追加する関数です。
⑤	tblnext "LAYER"	次の画層名を取得します。（nil の場合は終了）
⑥	nth 0 lst	画層名リストの 1 番目を取り出します。nth 関数は <0> から始まります。
⑦	open "out_layer.txt" "w"	ファイル変数 <out_layer.txt> を書き込みモードでオープンします。
⑧	setq i 1	書き出した画層数を カウントするために、初期値の <1> をセットします。
⑨	/= new_layer nil	リストに画層名がある限り、書き込みの処理を繰り返します。
⑩	write-line	1 行づつ順番に書き込みます。
⑪	nth i lst	リスト内の次の画層名を取得します。
⑫	setq i (1+ i)	ループ カウンタを <1> つ増やして、最初に戻ります。
⑬	close	最後は、必ずファイルを閉じます。
⑭	prompt	書き出した画層の数をコマンドラインに表示します。
	princ(- i 1)	new_layer が <nil> となった時点で、<i> は 1 つ多くカウントされているから、書き出した画層の数は <i> の数値から 1 つ少なくする。

AutoCAD と対話する

in_var.lsp	外部ファイルからシステム変数を読み込む
目的：	外部ファイル < テキスト ファイル > から、システム変数を読み込み、現在の図面のシステム変数を変更します。
主要関数：	<open><read_line><substr><atoi><osmode><close>

Step1 ー 外部ファイル <out_sys.txt> を読み込み <"r">、変数 <in_var> に代入します。❶

```
(setq  in_var  (open  "out_sys.txt"  "r"))
```

Step2 ーテキスト ファイルから、1 行づつ読み込み、システム変数名と数値に分離します。

```
(setq  ren_osmode (read-line  in_var))              → 1 行づつ読み込みます。❷
    (setq  *osmode_txt  (substr ren_osmode 1 6)) → 先頭から、6 文字を取り出します。❸
    (setq  *osmode_val (substr ren_osmode  7))    → 7 番目以降を取り出します。❹
    (setq  *osmode_val (atoi   *osmode_val))      → 文字列から数値に変更します。❺
(setq  ren_LUPREC (read-line  in_var)) ❻
    (setq  *LUPREC_txt  (substr ren_LUPREC 1 6)) ❼
    (setq  *LUPREC_val (substr ren_LUPREC 7)) ❽
    (setq  *LUPREC_val (atoi   *LUPREC_val)) ❾
```

→上（②〜⑤）を繰り返します。

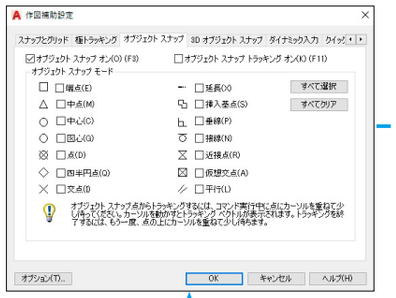
Step3 ー読み込んだシステム変数の値を現在のシステム変数値と入れ替えます。❿⓫

```
(setvar  "osmode"  *osmode_val)
(setvar  "LUPREC"  *LUPREC_val)
```

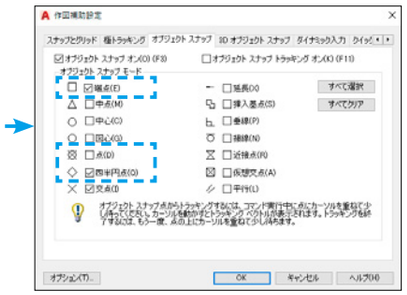
Step4 ーファイルを閉じます。（必須です）⓬

```
(close  in_var)
```

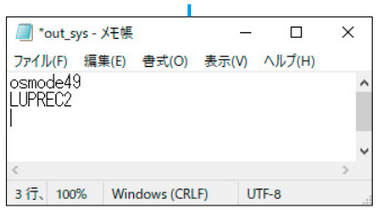
初期値は、OSMODE<0> です。



OSMODE<49> は [端点]+[四半円点]+[交点] です。



テキスト ファイルからシステム変数を読み込み、変更する。



Point!

```
(defun C:in_var ()
❶  (setq  in_var  (open  "out_sys.txt"  "r"))  ; ファイルを読み込みモードで開く   → Step1
;1 行ずつ読み込んで、システム変数名と設定値を変数に代入
❷  (setq  ren_osmode (read-line  in_var))  ;1 行ずつ読み込む
❸      (setq  *osmode_txt  (substr ren_osmode 1 6))  ; システム変数名を取得
❹      (setq  *osmode_val (substr ren_osmode  7))  ; システム変数の値を取得
❺      (setq  *osmode_val (atoi   *osmode_val))  ; 文字列を数値に変換
❻  (setq  ren_LUPREC (read-line  in_var))  ;1 行ずつ読み込む
❼      (setq  *LUPREC_txt  (substr ren_LUPREC 1 6))  ; システム変数名を取得
❽  (setq  *LUPREC_val (substr ren_LUPREC 7))  ; システム変数の値を取得
❾      (setq  *LUPREC_val (atoi   *LUPREC_val))  ; 文字列を数値に変換   → Step2
; ファイルから取得した数値でシステム変数をセットする
❿  (setvar  "osmode"  *osmode_val)  ;"osmode" を変更する
⓫  (setvar  "LUPREC"  *LUPREC_val)  ;"LUPREC" を変更する   → Step3

⓬  (close  in_var)  ; ファイルを閉じる   → Step4
  ) ;end
```

番号	関数名	説　明
①	open　"r"	外部のテキスト ファイルを読み込みモードで開きます。
②	read-line	開いたファイルから 1 行づつ読み込み、変数にセットします。
③	substr　1　6	読み込んだ文字列の先頭から 6 文字を取り出します。（システム変数名を取得します。）
④	substr　7	読み込んだ文字列の先頭から 7 文字目以降を取り出します。（システム変数の値を取得します。）
⑤	atoi	取り込んだシステム変数は文字列なので、整数に変換します。
⑥	read-line	次の 1 行を読み込みます。
⑦	substr　1 6	読み込んだ文字列の先頭から 6 文字を取り出します。（システム変数名を取得します。）
⑧	substr　7	読み込んだ文字列の先頭から 7 文字目以降を取り出します。（システム変数の値を取得します。）
⑨	atoi	取り込んだシステム変数は文字列なので、整数に変換します。
⑩	setvar	システム変数 [osmode] を変更します。
⑪	setvar	システム変数 [luprec] を変更します。
⑫	close	テキスト ファイルを閉じます。（必須）

Point!

```
(setq  in_var  (open  "out_sys.txt"  "r"))
```

❶ →ファイルの絶対パスを指定しない場合は AutoCAD の既定の図面フォルダになります。
ファイルが <C:autolisp> のフォルダにある場合は、C:¥¥autolisp¥¥ を付けます。

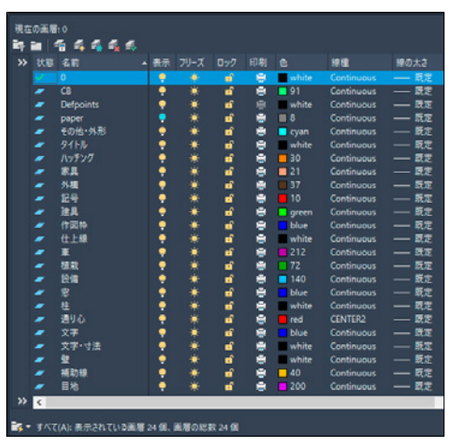
in_layer.lsp	外部ファイルから画層名一覧を読み込み、現在の図面に画層を追加する
目的：	外部ファイル < テキスト ファイル > から画層名一覧を読み込み、現在の図面に画層を追加します。
主要関数：	<open><read_line><list><while><append><close><nth><1+>

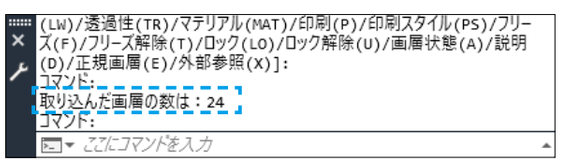
Step1 ー画層名の一覧を取り込みたい図面を開きます。

Step2 ー取り込みたい画層名ファイルがあることを確認します。

☞選択した画層だけを取り込む方法は P2-144 参考

Step3 ー（左図）in_layer.lsp を実行すると、全画層名が現在の図面に取り込まれます。
（右図）コマンドラインに取り込んだ画層の数を表示します。






```
(defun C:in_layer( / in_layer layer_name lst new_layer i)
; 外部にある画層名のテキスト ファイルを読み込む
❶ (setq  in_layer  (open "out_layer.txt" "r"))  ; 読み込み用のテキスト ファイルを開く
❷ (setq  layer_name (read-line in_layer))  ;1 行目を読み込む
❸ (setq  lst (LIST layer_name))  ; 読み込んだ画層名をリストにする準備（1 つ目をリストにする）
❹ (while (/= layer_name nil)  ;2 行目以降の処理（画層名が有れば、以下を実行する）
❺   (setq  layer_name (read-line  in_layer))  ; 次の行を読み込む
❻   (setq  lst (append lst (LIST layer_name)))  ; 取得した画層名をリストに追加する
   );while
❼ (close  in_layer)  ; ファイルを閉じる（必須）
; 画層を追加
❽ (setq new_layer (nth 0 lst))  ; リストの 1 番目を取り出す
❾ (setq i 1)  ; 読み込んだ画層数をコマンドラインに表示するためのカウントをセット
❿ (while (/= new_layer nil)  ; 画層名が無くなるまで読み込みを続ける
⓫   (command "-LAYER" "N" new_layer "")  ; 新しい画層を作成する
⓬   (setq new_layer (nth i lst))  ; リストの次の画層名をセットする
⓭   (setq i (1+ i))  ; 読み込んだ画層の数を <1> 増やして⑩から繰り返す
   );while
⓮ (prompt  "¥n 読み込んだ画層の数は：")  ; コマンドラインにメッセージを表示する
   (princ  (- i 1))(princ)  ; 画層の数を表示する
);end
```

番号	関数名	説　明
①	open "out_layer.txt" "r"	ファイル変数 <out_layer.txt> を読み込みモードでオープンします。
②	read-line	1 行づつ順番に読み込みます。
③	LIST layer_name	取り出した画層名を順番にリストにしていきます。
④	/= layer_name nil	リストに画層名がある限り、読み込みの処理を繰り返します。
⑤	read-line in_layer	1 行づつ順番に読み込みます。
⑥	append lst (LIST layer_name)	読み込んだ画層名を順番にリストにしていきます。
⑦	close	最後は、必ずファイルを閉じます。
⑧	nth 0 lst	画層名リストの 1 番目を取り出します。nth 関数は <0> から始まります。
⑨	setq i 1	読み込んだ画層数を カウントするために、初期値の <1> をセットします。
⑩	/= new_layer nil	リストに画層名がある限り、画層の新規作成を繰り返します。
⑪	"-LAYER" "N"	画層を新規作成します。"-LAYER" は画層のダイアログを表示しません。
⑫	nth i lst	リスト内の次の画層名を取得します。
⑬	setq i (1+ i)	ループ カウンタを <1> つ増やして、⑩に戻ります。
⑭	prompt	読み込んだ画層の数をコマンドラインに表示します。
	princ	new_layer が <nil> となった時点で、<i> は 1 つ多くカウントされているから、読み込んだ画層の数は <i> の数値から 1 つ少なくする。

☞①から⑦の解説は、P2-143 を参考

AutoCAD と対話する

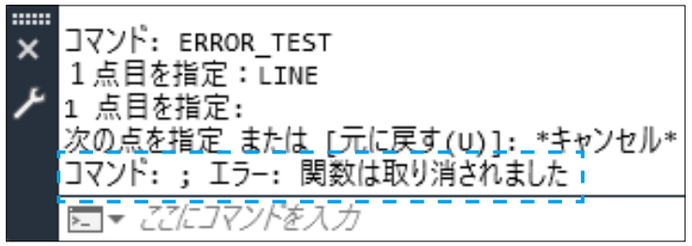
第 5 節 AutoLISP のエラー処理

AutoLISP には、エラーを処理するための関数がいくつかあります。これらの関数を使用すると、次のことが可能です。

①プログラム実行中にエラーが発生した情報をユーザーに提供します。

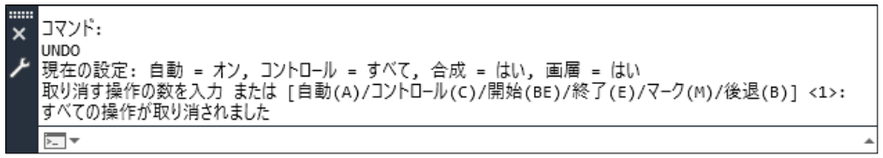
メッセージ ボックスに表示

コマンド ラインに表示



② AutoCAD の環境を元の状態に復元します。

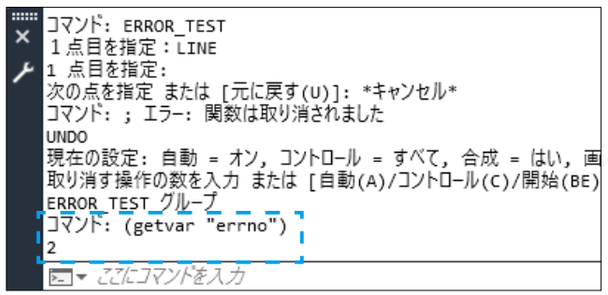
- [*error*] 関数を使用して、図面の設定などの条件をプログラム実行前に戻します。
- [undo] コマンドを実行して、すでに作図した図形を含めて、全てを元の状態に戻します。



③ AutoLISP の関数呼び出しによって発生したエラーを AutoCAD が検出したときに、そのエラーに該当するエラー コードの数値を表示できます。

- AutoLISP の (getvar "errno") 関数を使用して、ERRNO の現在値を検査します。

(setq a (getvar "errno")) → エラー番号が返されます。



使用頻度の高い AutoLISP のエラー処理関数を、次の表に示します。(順不同)

エラー処理関数	
関　数	説　明
(alert string)	文字列として渡されたエラー メッセージや警告メッセージを警告ダイアログ ボックスに表示します。
(*error* string)	ユーザー定義可能なエラー処理関数です。
(exit)	現在のアプリケーションを強制的に終了します。
(quit)	現在のアプリケーションを強制的に終了します。

1 AutoLISP のエラー コード

次の表に、AutoLISP で生成されるエラー コードの値を示します。AutoLISP の関数呼び出しによって発生したエラーを AutoCAD が検出すると、システム変数 ERRNO に次の値のいずれかが設定されます。(getvar "errno") を使用すると、AutoLISP のアプリケーションで現在の ERRNO の値を検査できます。

システム変数 [ERRNO] は、ゼロにクリアされない場合があります。AutoLISP の関数がエラーをレポートした直後に検査しないと、その値が示すエラーは誤っている可能性があります。
この変数は、図面を新規作成したり図面を開いたとき、必ずクリアされます。

ERRNO の値とその意味は、将来変更される可能性があります。

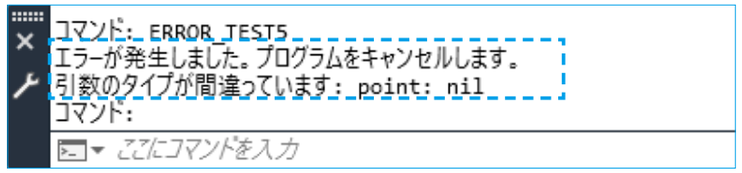
AutoLISP の エラー コード	
値	意　味
0	エラーはありません
1	無効なシンボル テーブル名です
2	無効な図形名または選択セット名です
3	選択セットの最大数を超えました
4	無効な選択セットです
5	ブロック定義が不正に使用されました
6	xref が不正に使用されました
7	オブジェクトを選択 : クリックが失敗しました
8	図形ファイルの終わりです
9	ブロック定義ファイルの終わりです
10	最後の図形が見つかりません
11	ビューポート オブジェクトを不正に削除しようとしました
12	PLINE[ポリライン] 中は操作できません
13	無効なハンドルです
14	ハンドルが使用可能になっていません
15	座標変換要求に無効な引数があります
16	座標変換要求に無効な空間があります
17	削除した図形が不正に使用されました
18	無効なテーブル名です
19	無効なテーブル関数の引数です
20	読み込み専用の変数に代入しようとしました
21	ゼロ値は許されていません
22	値が範囲外です
23	複雑な REGEN の処理中です
24	図形タイプを変更しようとしました
25	不正な画層名です
26	不正な線種名です
27	不正な色名です
28	不正な文字スタイル名です
29	不正なシェイプ名です
30	図形タイプのフィールドが不正です
31	削除した図形を変更しようとしました
32	Seqend 従属図形を変更しようとしました

33	ハンドルを変更しようとしました
34	ビューポートの可視性を変更しようとしました
35	図形はロックされている画層にあります
36	不正な図形タイプです
37	不正なポリライン図形です
38	ブロック内の複合化図形が不完全です
39	無効なブロック名フィールドです
40	ブロック フラグ フィールドが重複しています
41	ブロック名フィールドが重複しています
42	不正な法線ベクトルです
43	ブロック名がありません
44	ブロック フラグがありません
45	名前のないブロックが無効です
46	ブロック定義が無効です
47	必須フィールドがありません
48	拡張データ (XDATA) のタイプが認識できません
49	XDATA 内のリストのネストが正しくありません
50	APPID フィールドの位置が正しくありません
51	XDATA の最大サイズを超えました
52	オブジェクトを選択 : null 応答です
53	APPID が重複しています
54	ビューポート図形を作成または変更しようとしました
55	xref、xdef、xdep を作成または変更しようとしました
56	ssget フィルタ : リストが途中で終わっています
57	ssget フィルタ : テスト オペランドがありません
58	ssget フィルタ : opcode(-4) 文字列が無効です
59	ssget フィルタ : ネストが正しくないか条件条項が空です
60	ssget フィルタ : 条件条項の始まりと終わりが一致していません
61	ssget フィルタ : 条件条項における引数の数が誤っています (NOT または XOR)
62	ssget フィルタ : ネストの最大数を超えました
63	ssget フィルタ : グループ コードが無効です
64	ssget フィルタ : 文字列テストが無効です
65	ssget フィルタ : ベクトル テストが無効です
66	ssget フィルタ : 実数テストが無効です
67	ssget フィルタ : 整数テストが無効です
68	ディジタイザがタブレット モードではありません
69	タブレットが位置合わせされていません
70	タブレット引数が無効です
71	ADS エラー : 新規のリザルト バッファを割り当てられません
72	ADS エラー : null ポインタを検出しました
73	実行ファイルを開けません
74	アプリケーションは既にロードされています
75	最大数のアプリケーションが既にロードされています
76	アプリケーションを実行できません
77	互換性のないバージョン番号です
78	ネストしたアプリケーションをロード解除できません
79	アプリケーションがロード解除を拒否しました
80	アプリケーションが現在ロードされていません
81	メモリが不足しているのでアプリケーションをロードできません
82	ADS エラー : 変換マトリックスが無効です
83	ADS エラー : シンボル名が無効です
84	ADS エラー : シンボル値が無効です
85	ダイアログ ボックスの表示中は AutoLISP/ADS を操作できません

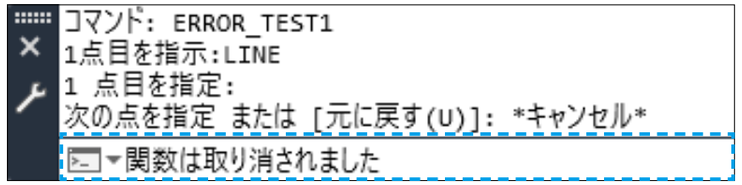
2 *error* 関数を使用する

ユーザーが *error* 関数を記述した場合、エラーが発生した後に AutoCAD を確実に特定の状態に戻すことができます。
このユーザー定義関数を使用すると、エラー状況に応じて適切なメッセージを表示します。
ユーザーが *error* 関数を記述していない場合、AutoCAD の操作中にエラーが生じると、
次の形式でメッセージを出力します。

① AutoLISP のコードに誤りがある場合の例。
コマンド ラインに次のように表示されます。
→コマンド：引数のタイプが間違っています : point: nil



②ユーザーが [Esc] キーを押して、キャンセルした場合の例。
コマンド ラインに次のように表示されます。
→ 関数は取り消されました

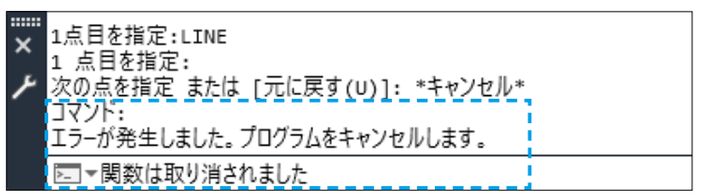


③このメッセージで、文字列 はエラーの内容を表します。しかし、*error* 関数が定義されている場合は 、AutoLISP はメッセージを表示しないで、*error* 関数を実行します。
error 関数は 文字列 を 1 つの引数として受け取ります。

④下記のように、ユーザー側で *error* 関数を作成し、AutoLISP のプログラムと一緒にロードするか、ACAD.lsp または ACADDOC.lsp に記述します。

```
(defun *error* (errmsg)
  (princ "¥n エラーが発生しました。プログラムをキャンセルします。")
  (terpri)
  (prompt errmsg)
  (princ)
)
```

エラーが発生した時、下図のように表示されます。



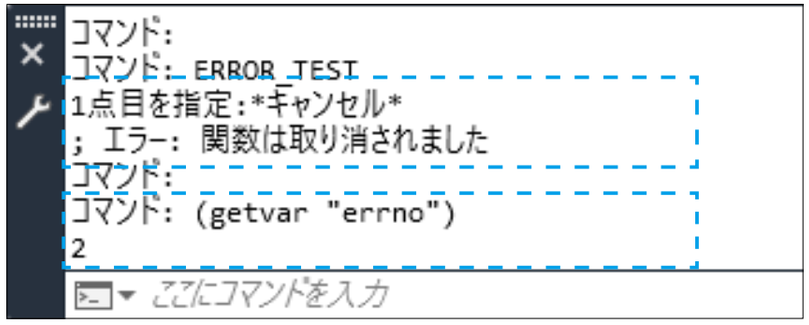
3 *error* 関数 でのエラー処理

次の LISP プログラムを使い、どのタイミングでエラーが生じるか見てみましょう。

```
          (defun  C:error_test ( )
 ①-------  (setvar  "OSMODE"  3)
 ②-------  (setq  pt1  (getpoint  "¥n1 点目を指示 :"))
 ③-------  (command  "LINE"   pt1  pause  "" )
 ④-------  (setvar  "OSMODE"  0)
          )
```

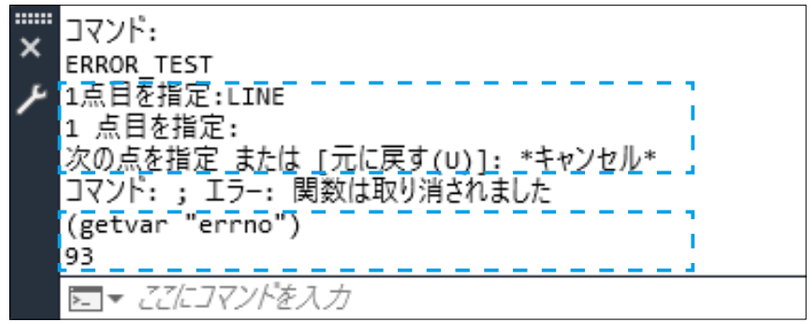
この LISP では、①の (setvar "OSMODE" 3) で、オブジェクト スナップ モードを <3> に設定しています。("OSMODE" 3　は、[端点 <1>]+[中点 <2>])
ユーザーが [Esc] キーを押して、プログラムをキャンセルするタイミングは、②と③です。
②の getpoint の get 関数は、ユーザーの指示を待つ関数です。
また、③の pause もユーザーの指示を待つ AutoCAD の関数です。

LISP の get 関数でエラーが生じた時のメッセージと AutoCAD のコマンド エラーのメッセージの違いを見てみましょう。
get 関数は、ユーザーの入力待ちの関数ですから、ほとんどの場合ここで LISP はストップします。
その時の AutoCAD のプロンプトには、一例として以下のようなメッセージが表示されます。



<1 点目を指示：> の場面で、Esc を押した時、コマンド ラインに <; エラー : 関数は取り消されました > のメッセージが出ます。(getvar "errno") で確認すると、<2> が表示されました。
エラーコード表で確認すると、<2> は < 無効な図形名または選択セット名です > であることがわかります。

次に、③の "LINE" の時に、Esc を押してエラー メッセージを確認してみます。



< 次の点を指定　または　[元に戻す (U)] : > の場面で、Esc を押した時、コマンド ラインに <; エラー : 関数は取り消されました > のメッセージが出ます。(getvar "errno") で確認すると、<93> が表示されました。<93> は AutoCAD の持つエラー メッセージです。
このように、ユーザー関数と AutoCAD の関数のエラー メッセージは違います。

上記の例で、ユーザーが Esc を押して、プログラムを中止したときに、問題になる点が 1 つあります。
それは、①で (setvar "OSMODE" 3) を行い、プログラムの最後の④で (setvar "OSMODE" 0) にしてオブジェクト スナップ モードを元に戻す処理をしていますが、②か③でプログラムを中止した場合、<"OSMODE" 3> の設定が残ってしまいます。

これを回避するために、AutoLISP には <*error*> 関数が用意されています。
記述方法は、(defun *error* (msg)) です。
この中の引数 (msg) には、エラー メッセージを引数から受け取りますが、このメッセージは <*error* 関数 > を使わないときに表示されるメッセージと同じで、AutoCAD から渡されます。

LISP のプログラムの最初に、このエラー関数を以下のように記述します。

```
(defun  *error*  (msg)
 (setvar  "osmode"   0)          → "osmode" を初期値 <0> に戻す処理
 (princ  msg)                    → エラー メッセージを表示する。
 (princ)
)
```

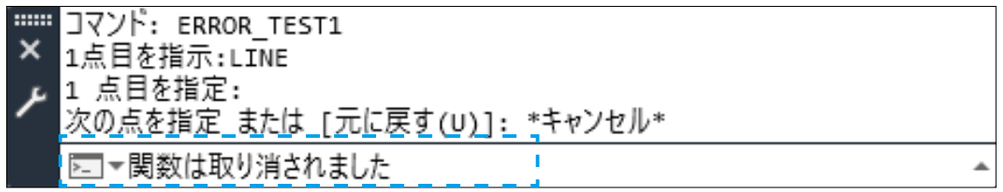
こうすれば、エラーで終了したときでも、変更した <osnap> などのシステム変数値は初期の設定に戻ります。

では、先ほどの LISP(error_test) に、このエラー関数を追加して確認します。

```
①-------(defun  *error*  (msg)
②-------  (setvar  "osmode"   0)
③-------  (princ  msg)
④-------  (princ)
          )
          (defun  C:error_test 1( )
            (setvar  "OSMODE"  3)
            (setq  pt1  (getpoint  "¥n1 点目を指示 :"))
            (command  "LINE"   pt1  pause  "")
            (setvar  "OSMODE"  0)
          )
```

①のエラー関数の定義で、AutoCAD のエラー メッセージを変数 <msg> に受け取ります。
次の②で <osnap> の設定を初期値の <0> に戻します。
次の③では、エラー メッセージをコマンドラインに表示します。
最後の④の <princ> は、同じメッセージを表示させないためです。

このエラー関数を追加したプログラムで、前回と同じように Esc でキャンセルした時のメッセージは次のようになります。



このように、コマンドの次に表示されるメッセージは、< 関数は取り消されました > だけが表示されます。（エラー：の文字は表示されません。）

エラー関数の中に、LISP の中で変更する予定のあるシステム変数を初期値に戻す処置を記述しておくことが大切です！

4 alert 関数 でエラー内容をダイアログで表示

[AutoCAD メッセージ] ボックスを表示して、ユーザーにエラー状況を知らせることもできます。
[AutoCAD メッセージ] ボックスは、プログラムが指定したメッセージを表示する小さなダイアログボックスです。[AutoCAD メッセージ] ボックスを表示するには、alert 関数を使用します。

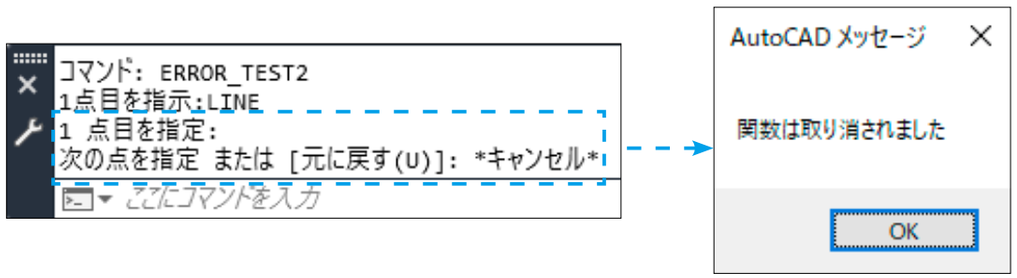
次のように alert 関数を呼び出すと、[AutoCAD メッセージ] ボックスを表示できます。
書式は (alert " メッセージ ") のように記述します。

では、前回のエラー関数を alert に置き換えた LISP を次に示します。

```
(defun  *error*  (msg)
  (setvar  "osmode"   0)        → エラーになった時に、システム変数を初期値に戻す
①----- (setq alert_msg msg)    → AutoCAD のエラーメッセージを変数 <alert_msg> に代入
②----- (alert alert_msg)        → 変数 <alert_msg> に代入したメッセージを表示する
  (princ)
)
(defun  C:error_test2 ( )
  (setvar  "OSMODE"  3)
  (setq  pt1  (getpoint  "¥n1 点目を指示 :"))
  (command  "LINE"   pt1  pause  "")
  (setvar  "OSMODE"  0)
)
```

ここで [Esc] を押すと、エラーメッセージが表示されます。

この LISP では、AutoCAD のエラー メッセージ <msg> を変数 <alert_msg> に代入して、その変数を alert 関数で表示させています。（コマンドラインに表示させるより、視覚的に判りやすい方法です。）

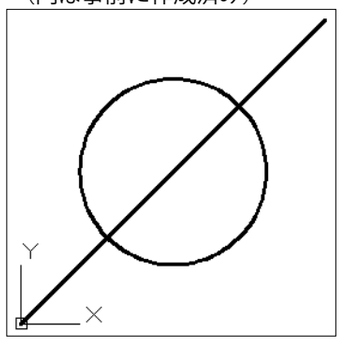


5 UNDO コマンド でエラーを処理する

前ページでは、[Esc] を押したときに元に戻す処理を行いましたが、下記の④のように⑤で [Esc] を押す前に、すでに線分を作図していれば、この線分は残ってしまいます。
これを回避するために、プログラム実行中にキャンセルした場合、[UNDO] コマンドを使って必ず最初に戻るような処理もできます。

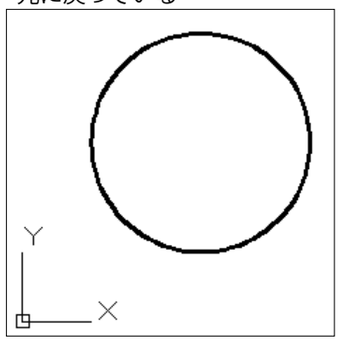
```
(defun error_undo (msg)
 (command-s "undo" "b" "y")      → [UNDO] コマンドの [B] で最初に戻す
 (setvar  "osmode"   0)          → エラーになった時に、システム変数を初期値に戻す
 (setq *error* errmsg)           → AutoCAD のエラーメッセージを元 (*error*) に戻す
 (setq alert_msg msg)            → AutoCAD のエラーメッセージを変数 <alert_msg> に代入
 (alert alert_msg)               → 変数 <alert_msg> に代入したメッセージを表示する
 (princ)
)

(defun  C:error_test3 ( / pt1)
①-----(setq errmsg *error*)      → AutoCAD のエラーメッセージを一時的に待避させる
②-----(setq *error* error_undo)  → AutoCAD の [*error*] に上記の補助関数をセットする
③-----(command "undo" "m")       → [UNDO] コマンドで [m]（マーク）をつける

④-----(command "line" '(0 0) '(100 100) "")  →線分を作図する
       (setvar  "OSMODE"  3)     →Oスナップモードを <3> に設定する
       (setq  pt1  (getpoint  "¥n1 点目を指示 :"))    ;1 点目の指示を促す
⑤-----(command  "LINE"   pt1  pause  "")      ┐→ここで [Esc] を押すと、エラーメッセージが
       (setvar  "OSMODE"  0)                  ┘  表示され、(error_undo) が実行されます。
      );end
```

④で線分は一旦は作図される
（円は事前に作成済み）



⑤でキャンセルされると、メッセージを表示して最初に戻る。

線分は削除され、OSMODE も元に戻っている



第 4 章 オブジェクトを操作する

前章までの LISP プログラムは図形の作図だけでしたが、実際の作図では図面の編集の作業も必要になります。
図面の編集をするには、図面情報が蓄えられているデータベースの情報を取得し、個々の図形 (オブジェクト) が持っている属性を変更します。

この章では以下のことを学びます。

第１節　オブジェクトの処理（単一図形）

図面内にある既存のオブジェクトの編集には以下のようなことが考えられます。

①オブジェクトのプロパティ（画層・色・線種など）を変更する。
②オブジェクトのジオメトリ（位置・大きさなど）を変更する。
③オブジェクトを削除したり、新規で作成する。

このようにオブジェクトを編集するには、そのオブジェクトの情報を取得し、その内容を理解しなければなりませが、これを手助けしてくれる関数が多くあります。
この節では、オブジェクトごとに編集する手法を学びます。

使用頻度の高い AutoLISP の オブジェクト処理関数を、次の表に示します。（順不同）

①オブジェクト処理関数（オブジェクトを取得）	
関　数	説　明
(entlast)	図面データベースの最後に追加されたエンティティ名を返します。
(entsel [msg])	メッセージ (msg) を表示してオブジェクトを選択し、そのオブジェクト名を最初の要素と選択に使った座標リストを 2 番目の要素とするリストを返します。メッセージ (msg) を省略すると、" オブジェクトを選択：" のプロンプトを表示します。
(nentsel [msg])	点を指定してオブジェクト (図形) を選択するようユーザに要求し、複合化オブジェクトに含まれる定義データにアクセスできるようにします。

②オブジェクト処理関数（オブジェクトのリストとエンティティ名を取得）	
関　数	説　明
(entget ename [applist])	エンティティ名で指定するオブジェクトの特性リストを返します。
(entnext [ename])	引数にエンティティ名を指定すると、エンティティ名を含む下位のエンティティの最初のエンティティ名を返します。 引数を省略すると、データベースの最初のエンティティ名を返します。

③オブジェクト処理関数 (オブジェクトの削除と作成）	
関　数	説　明
(entdel ename)	エンティティ名で指定するオブジェクトを削除します。 同じセッションで、すでに削除したものを指定すると、削除が取り消されます。
(entmake [elist])	図面内に新しいオブジェクトを作成します。
(entmakex [elist])	新しいオブジェクトを作成し、それにハンドルと図形名を与え、新しい図形名を返します。 (ただし、オーナーは割り当てません)

④オブジェクト処理関数 (オブジェクトの更新）	
関　数	説　明
(entmod [elist])	オブジェクトの定義データを更新します。
(entupd ename)	entmod で特性を変更したオブジェクトの表示を更新します。

1　オブジェクト処理関数（オブジェクトを取得する）

ENTLAST	最後のエンティティの名前を返します

(setq en1 (entlast))　→ 最後に作成した図形のデータを返します。

コマンド : !en1　→ コマンドラインに <!en1> と入力します。
< 図形名 : 7ef03560>　→ 図形の名前が表示されます。

entlast	最後に作図した情報を取得する

1. 最初に、適当な長さの線分を作図します。

2. コマンドラインから、以下のように入力します。
   (setq en1 (entlast))
   この式は、最後に作図した図形を取得します。選択する図形は 1 つです。
   この図形を変数 en1 にセットします。

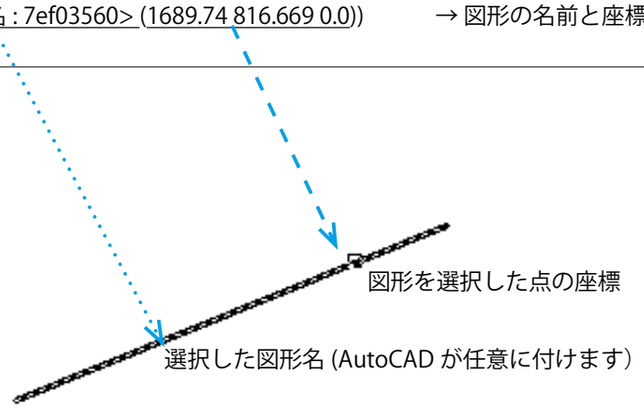
3. コマンドラインに　(< 図形名 : 7ef03990>) のような情報が表示されます。
   (数値は、 図形ごとに違います。)
   < 図形名 : 7ef03990> が選択した図形の名前です。1 つの図形には 1 つの名前が付けられます。

4. この図形の中身を表示する関数は、entget です。
   (setq en2 (entget en1)) と入力すると、以下のような情報が表示されます。
   コマンド : (setq en2 (entget en1))
   ((-1 . < 図形名 : 7ef03990>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2F6") (100 . "AcDbEntity")
   (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 1096.73 1249.12 0.0) (11 1255.82 1247.36 0.0)
   (210 0.0 0.0 1.0))

5. この情報から以下のことが判ります。
   (0 . "LINE")　→この図形は [線分]
   (410 . "Model")　→ Model 空間にある
   (8 . "0")　→画層は [0]
   (10 1096.73 1249.12 0.0)　→始点の座標 [1096.73 1249.12 0.0]
   (11 1255.82 1247.36 0.0)　→終点の座標 [1255.82 1247.36 0.0]
   (210 0.0 0.0 1.0)　→押し出し方向（既定値 =0,0,1）
   (X、Y、Z 座標)

ENTSEL	選択したエンティティ名と選択点の座標を返します

(setq en1 (entsel)) →キーボードから入力します。
オブジェクトを選択 : → 図形を選択します。
コマンド : !en1 → コマンドラインに <!en1> と入力します。
(< 図形名 : 7ef03560> (1689.74 816.669 0.0)) → 図形の名前と座標が表示されます。



entsel	図形情報を取得する

1. AutoCAD の線分 [LINE] コマンドで上記の線分を作図します。

2. コマンドラインから、以下のように入力します。
   (setq en1 (entsel))
   この式は、コマンドラインに < 図形選択 > の文字を表示して、ユーザーが図形を選択するのを待ちます。図形は 1 つしか選択できません。
   その図形を変数 en1 にセットします。

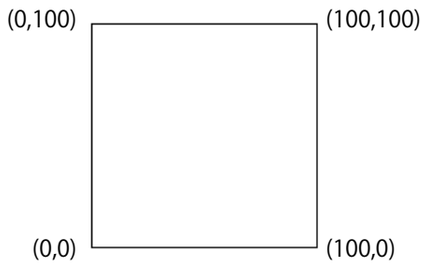
3. コマンドラインに (< 図形名 : 7ef03560> (1689.74 816.669 0.0)) のような文字と数値が表示されます。(上記の文字と数値は、図形ごとに必ず違います。)
   < 図形名 : 7ef03560> が選択した図形の名前です。1 つの図面には 1 つの名前が付けられます。
   (1689.74 816.669 0.0) は、マウスで図形を指示した位置の座標値です。

4. キーボードから、<!en1> と入力してみましょう。
   < 図形名 : 7ef03560> (1689.74 816.669 0.0)) の文字が返され、変数 en1 にこの線分の図形情報が取得されているのが分かります。

NENTSEL	従属図形 (頂点) の名前と指定した点の名前を含むリストを返します

(setq en1 (nentsel)) →キーボードから入力します。
オブジェクトを選択 : → 図形を選択します。
コマンド : !en1 → コマンドラインに <!en1> と入力します。
(< 図形名 : 7ef039a0> (18.1866 -0.577688 0.0)) → 図形の名前と指示した座標が表示されます。

ポリライン (PLINE) で作図した四角形



nentsel	ポリラインの図形情報を取得する

1. AutoCAD のポリライン [PLINE] コマンドで上記の正方形を作図します。

2. コマンドラインから、以下のように入力します。
   (setq en1 (entget (car (nentsel))))
   オブジェクトを選択すると以下のようなメッセージが表示されます。
   ((-1 . < 図形名 : 7ef039a0>) (0 . "LWPOLYLINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A4")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbPolyline") (90 . 4) (70 . 1) (43 . 0.0)
   (38 . 0.0) (39 . 0.0) (10 0.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (91 . 0) (10 100.0 0.0) (40 . 0.0) (41 . 0.0)
   (42 . 0.0) (91 . 0) (10 100.0 100.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (91 . 0) (10 0.0 100.0) (40 . 0.0) (41 . 0.0)
   (42 . 0.0) (91 . 0) (210 0.0 0.0 1.0))

3. ポリラインの開始頂点 (vertext) を示すドット・ペアのグループ コードは <10> ですが、上のリストには <10> のドット・ペアが 4 つあります。
   (10 0.0 0.0)、(10 100.0 0.0)、(10 100.0 100.0)、(10 0.0 100.0)

4. このように、PLINE は図面データベース内では単独の図形（複合図形ではない）として定義されています。ブロック図形や 3D ポリラインのような複合図形では上記と異なるリストが返ります。

[3D ポリライン] は、P1-219 を参考

entsel 関数で LWPOLYLINE の頂点座標を取得する

前ページで作図した正方形の 4 つの頂点座標を entsel 関数で取得します。

```
(defun C:get_LWPOLY ( / ent ent_len i ent1 )
❶ (setq ent (entget (car (entsel)))) ; 四角形の情報のリストを取得
❷ (setq ent_len (length ent)) ; 四角形の情報の数を取得 (35 個 )
❸ (setq i 0) ;nth 関数でドット・ペアを順番に取り出すためのカウンタを <0> にセット
❹ (repeat ent_len ; 括弧のリストの数だけ繰り返す
❺ (setq ent1 (car (nth i ent))) ; ドット・ペアのリストから前半のグループ コードを検査
❻ (if (= ent1 10) ; もし、グループ コードが <10> である時、以下の処理を行う
    (progn ;（グループ コード <10> が頂点座標であるため）
❼ (terpri) ; 改行する（結果を 1 行ずつずらして表示するため）
❽ (princ (cdr (nth i ent))) ;(10 . 頂点座標 ) の後半部分（頂点座標）だけを表示する
    );progn
  );if
❾ (setq i (1+ i)) ;nth 関数のカウンタを <1> 増やして、④に戻り次のドット・ペアを検査する
 );repeat
(princ)
);end
```

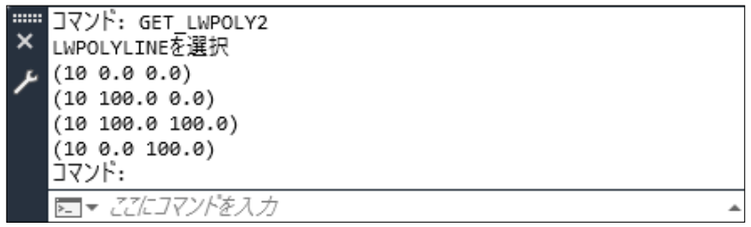
結果がコマンドラインに表示されます。
(0.0 0.0)
(100.0 0.0)
(100.0 100.0)
(0.0 100.0)

番号	関数名	説　明
①	(entget (car (entsel)))	(car (entsel)) で取得できるのは < 図形名 : 7ef03a10> なので、この中身を取得する関数が entget 関数です。
②	(length ent)	取得したドット･ペアの数を取得します。(repeat 関数で使用するため)
③	(setq i 0)	nth 関数のカウントを開始します。nth 関数は <0> から始まります。
④	repeat ent_len	取得したドット・ペアの数だけ繰り返します。
⑤	(car (nth i ent))	ドット・ペアの前半のグループ コードを取得します。(<10> が頂点座標)
⑥	if (= ent1 10)	グループ コードが <10> かどうかをチェックします。
⑦	terpri	改行する関数です。これが無いと、1 行に全部表示されます。
⑧	(cdr (nth i ent))	取得したドット・ペアの後半部分（座標値が有る）を取得します。
⑨	(setq i (1+ i))	カウントを <1> 加えて、④に戻ります。

nentsel 関数で LWPOLYLINE の頂点座標を取得する

P1-205 で作図した正方形の 4 つの頂点座標を nentsel 関数で取得します。

```
(defun C:get_LWPOLY2 ( / ent ed pt pt_list)
❶ (setq ent (nentsel "¥nLWPOLYLINE を選択 ")) ;LWPOLYLINE(PLINE) を選択する
❷ (setq ent_data (entget (car ent))) ; 前半の図形名を取得し、その情報を取得する
❸ (setq pt (assoc 10 ent_data)) ; 頂点座標リスト（グループコード <10>）を順番に取得する
; この図では、最初の pt は (0.0 0.0) になります。
❹ (while pt ;pt が存在する ( 真 ) 限り、以下のプログラムを実行する
❺ (print pt) ; 取得した pt の座標をコマンドラインに表示する。最初は (10 0.0 0.0)
; 取得した以降の頂点座標 (10 . 座標値 ) を取得します。
;member 関数は、初めに検出した式以降のリストを返します。
;1 番目の頂点座標 (10 0.0 0.0) を取得した後は、これ以降の頂点座標を探します。
❻ (setq ent_data (cdr (member pt ent_data))) ; これ以降のドット・ペアを再構成する
; 再度、ドット・ペアのグループから、頂点座標の座標値のみ取得します。
❼ (setq pt (assoc 10 ent_data)) ; 頂点座標のドット・ペア（グループコード <10>）を取得
;2 番目の場合は、(10 100.0 0.0) が取得できます。
 ); ④に戻って、pt が <nil> になるまで繰り返します。
(princ)
);end
```

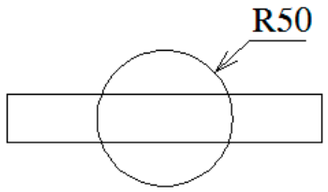


結果がコマンドラインに表示されます。
(10 0.0 0.0)
(10 100.0 0.0)
(10 100.0 100.0)
(10 0.0 100.0)

番号	関数名	説　明
①	nentsel	複合化されたオブジェクト（図形）の従属データを取得します。
②	(entget (car ent))	entget 関数でオブジェクトの中身を取得します。
③	(assoc 10 ent_data)	グループコードが <10> の最初のドット・ペアを取得します。
④	while pt	グループコードが <10> のドット･ペアが存在すれば以下を実行します。
⑤	(print pt)	検出したグループコードをコマンドラインに表示します。
⑥	(cdr (member pt ent_data))	検出したドット・ペア以降のドット・ペアを再構成します。残りのドット・ペアからグループコードが <10> のドット・ペアを探します。
⑦	(assoc 10 ent_data)	次のグループコードが <10> の最初のドット・ペアを取得します。

entsel & nentsel	ブロックの図形情報を取得する

下図はブロック図形（ブロック名は <light>）



entsel 関数でブロック図形を選択

① entsel 関数でブロックを選択します。

(setq ent1 (entget (car (entsel))))

→ ((-1 . < 図形名 : 7ef03ba8>) (0 . "INSERT") (330 . < 図形名 : 7ef01cf8>) (5 . "2E5")
(100 . AcDbEntity")(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbBlockReference") (2 . "light")
(10 1410.61 1368.26 0.0) (41 . 1.0) (42 . 1.0) (43 . 1.0) (50 . 0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0)
(210 0.0 0.01.0))

②上記のリストから判るのは、(0 . "INSERT") から図形はブロック、(2 . "light") からブロック名は "light"、挿入座標の (10 1410.61 1368.26 0.0) などです。

nentsel 関数でブロック図形の < 円 > を選択

① entsel 関数でブロックの < 円 > を選択します。

(setq ent1 (entget (car (nentsel))))

→ ((-1 . < 図形名 : 7ef03b98>) (0 . "CIRCLE") (330 . < 図形名 : 7ef03b80>) (5 . "2E3")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10 0.0 0.0 0.0) (40 . 50.0) (210 0.0 0.0 1.0))

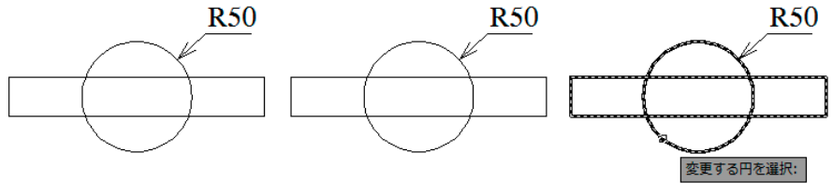
②上記のリストから判るのは、ブロック図形の中の円を選択すると、ブロックの情報ではなく円の情報が取得できます。(0 . "CIRCLE") から図形は円、(40 . 50.0) から半径が <50.0> と判ります。

Point!

ブロック図形を entsel 関数で選択すると、ブロック自体の情報を取得できますが、nentsel 関数で選択すると、ブロックを構成するオブジェクト（従属図形）の情報を取得します。
3D ポリラインも同様です。nentsel 関数で 3D ポリラインを選択すると、選択したセグメントの頂点座標（3D ポリラインの 2 番目のセグメントを選択すると、そのセグメントの頂点座標）を返します。

nentsel 関数でブロックを編集する

前ページで作図したブロック図形 (light) の円の大きさを <50> から <100> に変更します。



① nentsel 関数で、右端のブロックの中の円を選択します。（同じブロックが 3 つあります。）

(setq ent1 (nentsel "¥n 変更する円を選択 "))

→ (< 図形名 : 7ef02ad8> (1883.46 1332.43 0.0) ((1.0 0.0 0.0) (0.0 1.0 0.0) (0.0 0.0 1.0)
(1921.85 1368.26 0.0)) (< 図形名 : 7ef02b48>))

②前半の図形名を取得し、その情報を取得します。

(setq ent_data (entget (car ent1)))

→ ((-1 . < 図形名 : 7ef02ad8>) (0 . "CIRCLE") (330 . < 図形名 : 7ef02ac0>) (5 . "2E3")
(100 . "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10 0.0 0.0 0.0) (40 . 50.0) (210 0.0 0.0 1.0))

③上記の半径のドット・ペア (40 . 50.0) を取得します。

(setq ent_data (assoc 40 ent1))

→ (40 . 50.0)

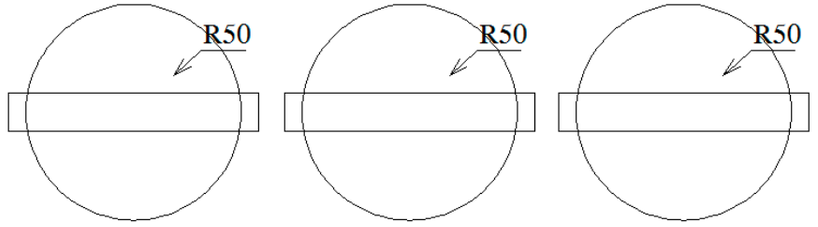
④新しい半径 (40 . 100.0) と入れ替えます。

(setq ent1 (subst '(40 . 100.0) ent_data ent1))

→ ((-1 . < 図形名 : 7ef02ad8>) (0 . "CIRCLE") (330 . < 図形名 : 7ef02ac0>) (5 . "2E3")
(100. "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10 0.0 0.0 0.0) (40 . 100.0) (210 0.0 0.0 1.0))

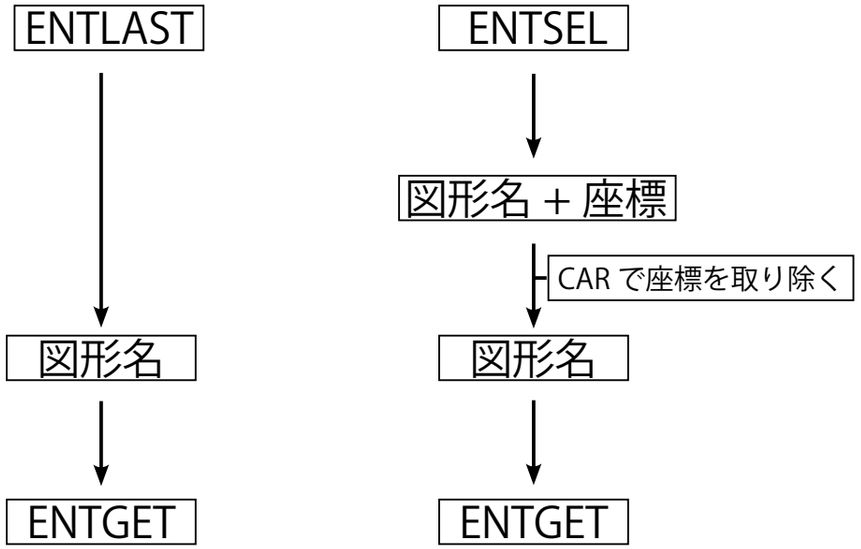
⑤ブロック図形の情報を更新します。下図のように他の同じブロックも同時に更新されました。

(entmod ent1)



entlast と entsel の相違

entlast 関数は最後に作図した図形の名前が取得できますが、entsel 関数は図形名と座標のドット・ペアの形で取得しますから、それから car 関数で第一番目の図形名を取得する過程が加わります。



データ項目を構成するグループ（代表的な例）

エンティティタイプ	データの記入項目
LINE （線分）	10，20，30　（始点） 11，21，31　（終点）
CIRCLE （円）	10，20，30　（中心点） 40　（半径）
ARC （円弧）	10，20，30　（中心点） 40　（半径） 50　（始点での角度） 51　（終点での角度）
TEXT （1 行文字）	10，20，30　（挿入点） 40　（高さ） 1　（文字の値） 50　（文字の回転角度） 7　（字体名） 11，21，31　（位置合わせの座標）
BLOCK （複合図形）	2　（複合図形名） 70　（複合図形のタイプ フラグ） 3　（複合図形名） 10，20，30　（複合図形の基点）
INSERT （図形挿入）	2　（複合図形名） 10，20，30　（挿入基点） 41　（X 方向の尺度） 42　（Y 方向の尺度） 43　（Z 方向の尺度） 50　（回転角度）

2 オブジェクト処理関数（オブジェクトのリストとエンティティ名を取得する）

ENTGET	指定したエンティティのデータのリストを返します

(setq en1 (entget (entlast)))　→ 最後の図形のエンティティ データを取り出します。

コマンド : !en1

((-1 . < 図形名 : 7ef03568>) (0 . "ARC") (330 . < 図形名 : 7ef01cf8>) (5 . "225") (100 . "AcDbEntity")
(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 1439.74 1568.78 0.0) (40 . 248.165)
(210 0.0 0.0 1.0) (100 . "AcDbArc") (50 . 0.434579) (51 . 2.23834))

上記のリストから読み取れる主な情報は、

(0 . "ARC")　→ 図形は円弧
(410 . "Model")　→ 図形はモデル空間にある
(8 . "0")　→ 図形の画層は <0>
(10 1439.74 1568.78 0.0)　→ 中心座標
(40 . 248.165)　→ 半径
(50 . 0.434579)　→ 開始角度（ラジアン）
(51 . 2.23834)　→ 終了角度（ラジアン）

AutoCAD エンティティのグループ コード

グループ コード	意　味
0	エンティティ タイプ（"LINE" や "CIRCLE" など）
1	最初の文字の値
2	名前（複合図形や属性など）
3 ～ 4	その他の文字や名前の値
5	16 進数文字列として表される図形ハンドラ
6	線種名
7	字体名
8	画層名
9	変数名
10	最初の X 座標（線分の始点、円の中心など）
11 ～ 18	その他の X 座標
20	最初の Y 座標
21 ～ 28	その他の Y 座標
30	最初の Z 座標
31 ～ 37	その他の Z 座標
38	図形の高度
39	図形の厚さ
40 ～ 48	浮動小数点値
49	反復される値
50 ～ 58	角度
62	色番号

entget	取得した図形の中身を見る

1. P204 の変数 en1 は、線分の図形情報と指示した位置の座標情報の組み合わせですが、知りたいのは図形情報ですから、この 2 つの組み合わせから第 1 番目の図形情報を取り出します。
   第 1 番目の情報を取り出す関数は <car> です。
   (car en1)

2. 取り出した図形の中身を表示する関数 <entget> を使い、その情報を変数 en2 に代入します。
   (setq en2 (entget (car en1)))

3. コマンドラインから、!en2 と入力してみましょう。下記のような数値が表示されます。
   ((-1 . <Entity name: 3b1e058>) (0 . "LINE") (5 . "32") (67 . 0) (8 . "0") (62 . 256) (6 . "BYLAYER")
   (48 . 1.00000) (60 . 0) (39 . 0.000000) (10 2.54225 5.60870 0.000000) (11 6.32486 6.96196 0.000000)
   (210 0.000000 0.000000 1.00000))

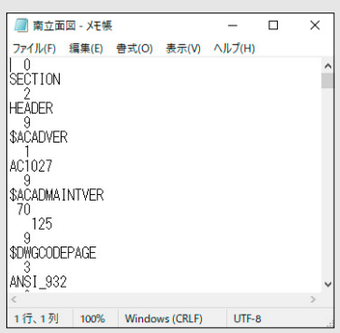
上記の数字は、1 つの例になりますが、これがこの線分の図形要素のリストになります。
重要な個所だけ見てみましょう。

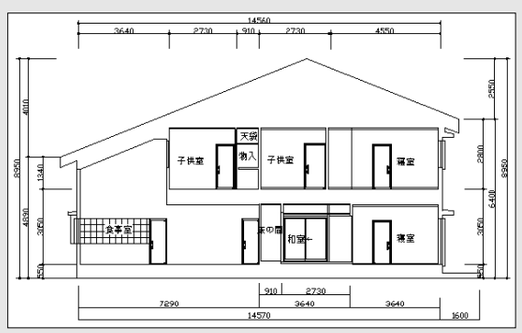
((-1 . <Entity name: 3b1e058>)	・・・	<-1> 図形要素名
(0 . "LINE")	・・・	<0> データタイプ (この場合は" 線分")
(8 . "0")	・・・	<8> 画層名 (この場合は" 0")
(10 2.54225 5.60870 0.000000)	・・・	<10> 線分の始点の座標
(11 6.32486 6.96196 0.000000)	・・・	<11> 線分の終点の座標

それぞれのリストは、2 つの項目からなっています。
最初の部分は、図形要素リストの項目を示すグループコード番号です。
グループコードは DXF のグループコードと同じです。

Point!

左のテキストファイルは、右の [南立面図 .dwg] を DXF 形式で保存したテキストです。
図形のグループコードは DXF のグループコードと同義です。(巻末の DXF グループコード表を参照)

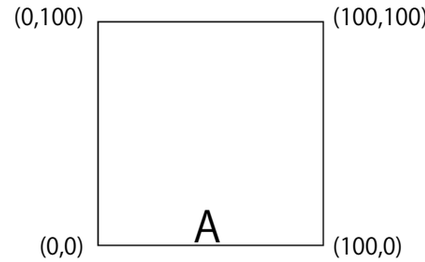




ENTNEXT	エンティティ名を含む下位のエンティティの最初のエンティティ名を戻します。

(setq en1 (entnext)) →キーボードから入力します。
コマンド : !en1 → コマンドラインに <!en1> と入力します。
< 図形名 : 7ef02a08> →図面の最初の 図形の名前が表示されます。
(setq en2 (entnext en1)) → en1 の次の図形を取得します。

3D ポリライン (3DPOLY) で作図した四角形の頂点座標を取得する



P1-205 の PLINE とこのページの 3D ポリラインは異なります。
PLINE（ポリライン）コマンドで作成された図形は [LWPOLYLINE] です。

3D ポリラインを entsel 関数で選択

① entsel 関数で 3D ポリラインを選択します。
(setq en1 (entget (car (entsel))))
→ ((-1 . < 図形名 : 7ef02d08>) (0 . "POLYLINE") (330 . < 図形名 : 7ef01cf8>) (5 . "321")
(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 ."AcDb3dPolyline") (66 . 1) (10 0.0 0.0 0.0)
(70 . 9) (40 . 0.0) (41 . 0.0) (2100.0 0.0 1.0) (71 . 0) (72 . 0) (73 . 0) (74 . 0) (75 . 0))

②上記のリストから判るのは、(0 . "POLYLINE") から図形はポリラインですが、頂点座標 (10 のドット・ペア) が見つかりません。

3D ポリラインを nentsel 関数で選択

① nentsel 関数で 3D ポリラインの下辺 A を選択します。
(setq en1 (entget (car (nentsel))))
→ ((-1 . < 図形名 : 7ef02d18>) (0 . "VERTEX") (330 . < 図形名 : 7ef02d08>) (5 . "323")
(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 ."AcDbVertex")
(100 . "AcDb3dPolylineVertex") (10 0.0 0.0 0.0) (40 . 0.0) (41 .0.0) (42 . 0.0) (70 . 32) (50 . 0.0) (71 . 0)
(72 . 0) (73 . 0) (74 . 0))

②上記のリストから判るのは、(0 . "VERTEX") と頂点座標の (10 0.0 0.0 0.0) です。
nentsel 関数で 3D ポリラインを選択すると、選択したセグメントの頂点座標（3D ポリラインの 1 番目のセグメントを選択すると、そのセグメントの頂点座標）が返ります。

entnext 関数で 3D ポリラインの頂点座標を取得する

前ページで作図した正方形の 4 つの頂点座標を nentsel 関数で取得しようとすると、3D ポリラインの全てのセグメントを選択しなければなりません。
しかし、entsel 関数と entnext 関数を使うと、頂点座標は取得できます。

① entsel 関数で 3D ポリラインを選択します。

```
(setq  ent (entget (car (entsel))))
→ ((-1 . < 図形名 : 7ef03958>) (0 . "POLYLINE") (330 . < 図形名 : 7ef01cf8>) (5. "29B")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDb3dPolyline") (66 . 1) (10 0.0 0.0 0.0)
   (70 . 9) (40 . 0.0) (41 . 0.0) (210 . 0.0 0.0 1.0) (71 . 0) (72 . 0) (73 . 0) (74 . 0) (75 . 0))
```

② ent の従属図形 (ent1) を entnext 関数で取得します。

```
(setq  ent1 (entget (entnext  (cdr  (car ent)))))
→ ((-1 . < 図形名 : 7ef03968>) (0 . "VERTEX") (330 . < 図形名 : 7ef03958>) (5 . "29D")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbVertex")
   (100 . "AcDb3dPolylineVertex") (10 0.0 0.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 32) (50 . 0.0) (71 . 0)
   (72 . 0) (73 . 0) (74 . 0))
```

③ ent1 の従属図形 (ent2) を entnext 関数で取得します。

```
(setq  ent2 (entget (entnext  (cdr  (car ent1)))))
→ ((-1 . < 図形名 : 7ef03970>) (0 . "VERTEX") (330 . < 図形名 : 7ef03958>) (5 . "29E")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbVertex")
   (100 ."AcDb3dPolylineVertex") (10 100.0 0.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 32) (50 . 0.0)
   (71 . 0) (72 . 0) (73 . 0) (74 . 0))
```

④ ent2 の従属図形 (ent3) を entnext 関数で取得します。

```
(setq  ent3 (entget (entnext  (cdr  (car ent2)))))
→ ((-1 . < 図形名 : 7ef03978>) (0 . "VERTEX") (330 . < 図形名 : 7ef03958>) (5 . "29F")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbVertex")
   (100 ."AcDb3dPolylineVertex") (10 100.0 100.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0)(70 . 32) (50 . 0.0)
  (71 . 0) (72 . 0) (73 . 0) (74 . 0))
```

⑤ ent3 の従属図形 (ent4) を entnext 関数で取得します。

```
(setq  ent4 (entget (entnext  (cdr  (car ent3)))))
→ ((-1 . < 図形名 : 7ef03980>) (0 . "VERTEX") (330 . < 図形名 : 7ef03958>) (5 . "2A0")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbVertex")
   (100 . "AcDb3dPolylineVertex") (10 0.0 100.0 0.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (70 . 32) (50 . 0.0)
   (71 . 0) (72 . 0) (73 . 0) (74 . 0))
```


⑥ ent4 の従属図形 (ent5) を entnext 関数で取得します。

```
(setq  ent5 (entget (entnext  (cdr  (car ent4)))))
→ ((-1 . < 図形名 : 7ef03960>) (0 . "SEQEND") (330 . < 図形名 : 7ef03958>) (5 . "29C")
   (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (-2 . < 図形名 : 7ef03958>))
```

⑦上記①から⑥までの図形名を抜き出すと以下の通りです。

```
ent    →   (0 . "POLYLINE")
ent1   →   (0 . "VERTEX")    頂点座標は (10 0.0 0.0 0.0)
ent2   →   (0 . "VERTEX")    頂点座標は (10 100.0 0.0 0.0)
ent3   →   (0 . "VERTEX")    頂点座標は (10 100.0 100.0 0.0)
ent4   →   (0 . "VERTEX")    頂点座標は (10 0.0 100.0 0.0)
ent5   →   (0 . "SEQEND")
```

⑧ "SEQEND" は、[シーケンス終了] の図形であることを示しています。このように (0 . "SEQEND") が表れるまで、entnext 関数で従属図形を検索していきます。

⑨下記は①から⑧をコード化したものです。

```
(defun c:get_POLYLINE ( / ent  i)
❶       (setq ent (entget (car (entsel))))  ; 四角形の情報のリストを取得
❷       (setq i 1)  ;"SEQEND" が見つかった時に、i を nil にセットしてループを終了する
❸       (while i  ; ループを開始する
❹       (setq ent (entget (entnext (cdr (car ent)))))  ; 選択した主図形の従属図形を取得
❺       (if (/= (cdr (assoc 0 ent)) "SEQEND")  ; グループ コードが <0> のドット・ペアの後半が
                (progn                         ;"SEQEND" でなければ、以下を行う
❻               (terpri)                       ; 改行する（結果を 1 行ずつずらして表示するため）
❼               (princ (cdr (assoc 10 ent)))   ;(10 . 頂点座標 ) の後半部分だけを表示する
                );progn
❽               (setq i nil)                   ; 最後の表示が終わると、ループを終了する
            );if
        );while
    (princ)
);end
```

⑩実行すると、コマンドラインに結果が表示されます。

```
オブジェクトを選択 :
(0.0 0.0 0.0)
(100.0 0.0 0.0)
(100.0 100.0 0.0)
(0.0 100.0 0.0)
```

3 オブジェクト処理関数 (オブジェクトの削除と作成)

ENTDEL	エンティティ名で指定した図形を削除します

(entdel (car en1)) → エンティティ名 < 7ef03560 > の図形は削除されます。

entdel の引数は、エンティティ名 < 7ef03560 > だけですから、<en1> から座標値を取り除いた前半部分だけを取り出します。(リストの第 1 番目を取り出す関数は、<car> です。)

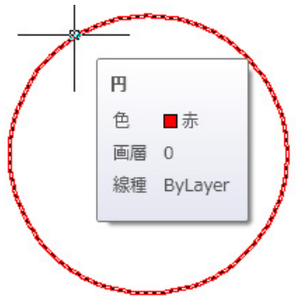
ENTMAKE	図面内に新しい図形を作成します

書式 : (entmake [elist])
注 : 引数の elist
entget 関数が返すのと同じ形式の図形定義データのリストです。elist 引数には、図形の定義に必要な情報がすべて含まれていなければなりません。必要な定義データが抜けていると、entmake 関数は nil を返し、その図形は無効になります。省略可能な定義データ (画層など) を省略した場合、entmake 関数は既定値を使用します。

下記のコードは座標 (10,20) を中心とする半径 1 00 の赤い円 (色 62<1>) を作成します。
省略可能な画層と線種のフィールドは省略されているので、既定値とみなされます。

コマンド : (entmake '((0 . "CIRCLE") (62 . 1) (10 10.0 20.0 0.0) (40 . 100.0)))
コマンドラインには、以下のように図形の定義データのリストが返ります。
((0 . "CIRCLE") (62 . 1) (10 10.0 20.0 0.0) (40 . 100.0))

画層 <0> に作図された赤い円



entget 関数で確認すると、以下のように表示されます。
指定した以外の情報は、自動的に付加されています。

コマンド : (setq en1 (entget(entlast)))
((-1 . < 図形名 : 7ef039e8>) (0 . "CIRCLE") (330 . < 図形名 : 7ef01cf8>) (5 . "2AD") (100
. "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (62 . 1) (100 .
"AcDbCircle") (10 10.0 20.0 0.0) (40 . 100.0) (210 0.0 0.0 1.0))

円のグループ コード	
エンティティタイプ	データの記入項目
CIRCLE (円)	8 (画層) 10 (中心点) 40 (半径) 62 (色)

4 オブジェクト処理関数 (オブジェクトの更新)

ENTMOD	エンティティ リストのデータベースを更新します

(setq en1 (entget (entlast))) → 最後の図形のエンティティ データを取り出します。
(setq en1 (subst (cons 8 "1") (assoc 8 en1))) → 画層 <1> と入れ替えます。
(entmod en1) → entmod 関数で、データベースを更新します。

コマンド : !en1
((-1 . < 図形名 : 7ef03550>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "222") (100 .
"AcDbEntity") (67 . 0) (410 . "Model") (8 . "1") (100 . "AcDbLine") (10 1403.87
1390.95 0.0) (11 1816.94 1540.2 0.0) (210 0.0 0.0 1.0))

上記のリストから、画層が <1> に変更されたことが判ります。
(8 . "1") → (8 . "1") の <8> は、画層を示すグループ コードです。
注 : 画層 <1> が存在していなければなりません。

図形の更新までの流れ

Step1 ー entlast 関数で最後の図形を取得します。
(setq en1 (entlast))

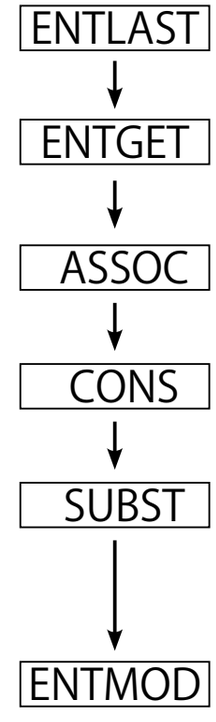
Step2 ー entget 関数で図形情報を取得します。
(setq en1 (entget en1))

Step3 ー assoc 関数で en1 の画層情報を取得します。
(assoc 8 en1)

Step4 ー cons 関数で、画層名 <1> のドット・ペアを作成します。
(cons 8 "1")

Step5 ー subst 関数で古い画層名と新しい画層名 <1> を入れ替えます。
(subst (cons 8 "1") (assoc 8 en1))

Step6 ー entmod 関数で、図形のデータベースを更新します。
(entmod en1)



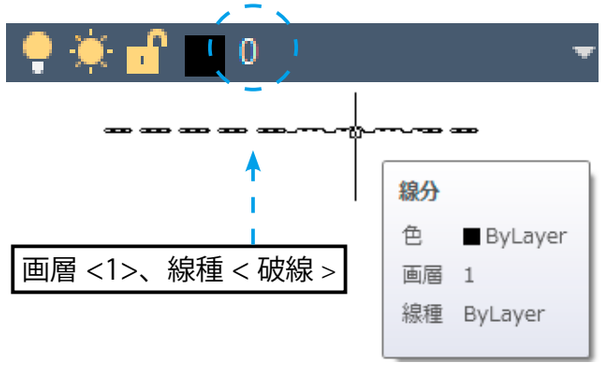
オブジェクトの操作

ChgLay.lsp	現在層を指示した図形の画層に変更する
目的：	現在層を選択した図形の画層に変更します。例えば、現在の画層が <0> のときに、選択した図形の画層が <1> であれば、現在層が <1> に変更されます。
主要関数：	<entsel><car><entget><assoc><cdr><LAYER>

Step1 ー現在の画層が <0> の時、画層 <1> の図形（線分）を選択します。❶

コマンドラインに " 現在層にしたい図形を選択：" を表示させて、現在層に変更したいオブジェクトを選択します。

下図では、現在層は <0> ですが、画層 <1> の線分を指示して画層を <1> に変更します。



Step2 ー変数 <ent1> には下記の情報が取得できます。

(< 図形名 : 7ef07b28> (918.647 1174.55 0.0))

① 1 番目のリストが図形名ですから、car 関数で取得します。❷

(car ent1) → < 図形名 : 7ef07b28>

②図形名の中身を展開する関数 entget を使います。❷

(entget (car ent1))

((-1 . < 図形名 : 7ef07b28>) (0 . "LINE") (330 . < 図形名 : 7ef06cf8>) (5 . "2E5") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "1") (100 . "AcDbLine") (10 912.822 1174.55 0.0) (11 929.785 1174.55 0.0) (210 0.0 0.0 1.0))

③画層名のあるドットペアは、(8 . "1") ですから、これから選択したオブジェクトの画層は "1" であることが判ります。

④画層のドットペア (8 . "1") を、リストから取り出します。❸

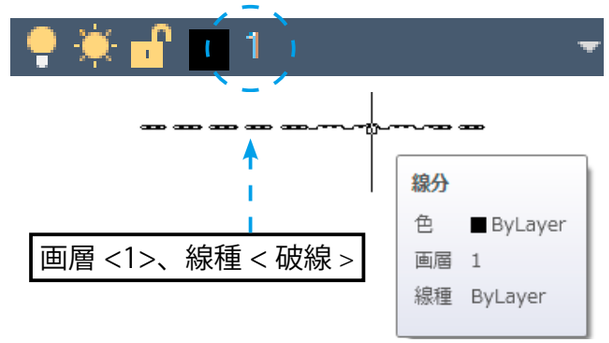
(assoc 8 ent2) → (8 . "1")

⑤これから、" 1" の文字だけ取得するには、リストの 2 番目以降の情報を取得する cdr 関数を使います。❹

(cdr ent3) → "1"

Step3 ー画層コマンドで現在層を <0> から <1> に変更します。❺

現在の画層が <1> に変更されました。




```
(defun C:ChgLay( / ent1 ent2 ent3 ent4)
;選択したオブジェクトの画層に変更します。
❶ (setq ent1 (entsel "¥n 現在層にしたい図形を選択：" ))          → Step1
;オブジェクトの画層名を取得します。
❷ (setq ent2 (entget (car ent1)))  ;図形の情報を取得
❸ (setq ent3 (assoc 8 ent2))       ;図形の画層のドット・ペアを取得   → Step2
❹ (setq ent4 (cdr ent3))           ;画層名を取得
;画層を変更します。
❺ (command "LAYER" "S" ent4 "" )   ;画層を変更                    → Step3
) ;end
```

番号	関数名	説　明
①	entsel	ユーザーがマウスで選択した図形 (図形名と座標) を変数 ent1 にセットします。 entsel の後に " と " に文字を入力すると、コマンドラインに表示されます。 文字を省いた場合は、< 図形を指示 > のメッセージが表示されます。
②	entget	ent1 の中には、図形名と座標値が入っていますが、必要なのは図形名ですから car 関数で第 1 番目の引数 (図形名) を取得しています。
③	assoc	図形要素リストから画層の項目だけ取り出します。(8 . 画層名)
④	cdr	画層名だけを取り出します。（cdr 関数は、2 番目以降の要素を取り出します）
⑤	layer	画層コマンドを使って、現在の画層を ent4 の画層 ("1") に変更します。 (S はレイヤー変更のオプション、 "" は画層コマンドの終了)

画層のオプションには、以下の項目があります。（この中の < 現在の層の変更 (S)> を使います）

オプションを入力

[一覧 (?)/ 現在の層の新規作成 (M)/ 現在の層の変更 (S)/ 新規作成 (N)/ 名前変更 (R)/ 表示 (ON)/ 非表示 (OF)/ 色設定 (C)/ 線種設定 (L)/ 線の太さ (LW)/ 透過性 (TR)/ マテリアル (MAT)/ 印刷 (P)/ フリーズ (F)/ フリーズ解除 (T)/ ロック (LO)/ ロック解除 (U)/ 画層状態 (A)/ 説明 (D)/ 正規画層 (E)]:

LayChg.lsp	選択した図形を現在層に変更する
目的：	選択した図形の画層を現在層に移します。例えば、選択した図形の画層 <1> を現在の画層 <0> に変更します。
主要関数：	<getvar><cons><entsel><entget><assoc><subst><entmod>

Step1 ー①現在の画層名を変数 <C_lay> に取得しておきます。❶

(getvar “CLAYER”) → 現在層の <0> が取得されます。

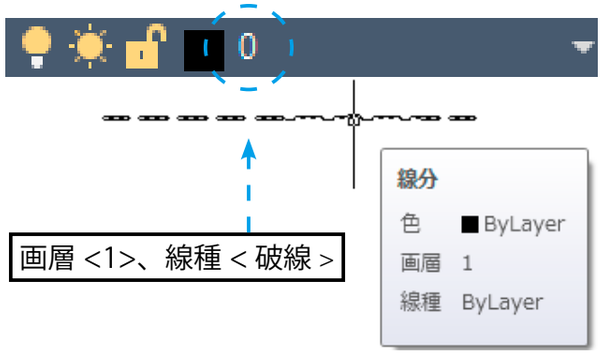
②この画層名のドットペアを作成します。ドットペアを作成する関数は、cons です。❷

(cons 8 C_lay) → (8 . "0")

Step2 ー現在の画層が <0> の時、画層 <1> の図形（線分）を選択します。❸

コマンドラインに " 現在層に移したい図形を選択：” を表示させて、現在層に移すオブジェクトを選択します。

下図では、選択したオブジェクトの画層は <1> ですが、現在層 <0> に移します。



Step3 ー変数 <ent1> には下記の情報が取得できます。

(< 図形名 : 7ef07b28> (924.267 1174.44 0.0))

① 1 番目のリストが図形名ですから、car 関数で取得します。

(car ent1) → < 図形名 : 7ef07b28>

②図形名の中身を展開する関数 entget を使います。❹

(entget (car ent1))

((-1 . < 図形名 : 7ef07b28>) (0 . "LINE") (330 . < 図形名 : 7ef06cf8>) (5 . "2E5") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "1") (100 . "AcDbLine") (10 912.822 1174.55 0.0) (11 929.785 1174.55 0.0) (210 0.0 0.0 1.0))

③画層名のあるドットペアは、(8 . "1") ですから、これから選択したオブジェクトの画層は "1" であることが判ります。

④画層のドットペア (8 . "1") を、リストから取り出し、変数 <Old_lay> にセットします。❺

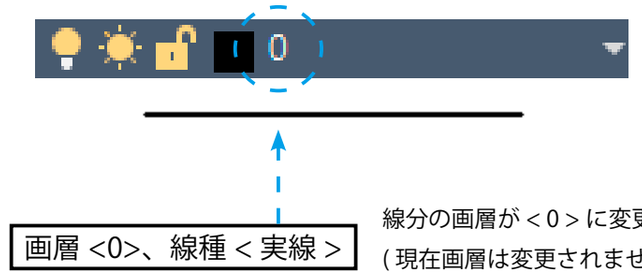
(assoc 8 ent2) → (8 . "1")

Step4 ー① Step1 で作成した画層のドットペアと Step3 で取得したドットペアを入れ替える関数は subst です。書式は、(subst 新しいドットペア 古いドットペア 図形名) です。❻

(subst New_lay Old_lay ent1))

② subst 関数で入れ替えた後は、必ず entmod 関数で図形情報を更新します。❼

書式は、(entmod 図形名) です。



線分の画層が < 0 > に変更されました。
(現在画層は変更されません。)

```
(defun C:LayChg( / C_lay New_lay ent1 ent2 Old_lay)
; 現在の画層名を取得し、ドットペアを作成します。
❶ (setq C_lay (getvar “CLAYER” )) ; 現在の画層名を取得する
❷ (setq New_lay (cons 8 C_lay)) ; 新しい画層名のドット・ペアを作成        → Step1
; 現在層に移したいオブジェクトを選択します。
❸ (setq ent1 (entsel “¥n 現在層に移したい図形を選択：” ))                  → Step2
; 選択したオブジェクトの画層名のドットペアを取得します。
❹ (setq ent2 (entget (car ent1))) ; 選択した図形の情報を取得する
❺ (setq Old_lay (assoc 8 ent2)) ; 画層名のドット・ペアを取得する           → Step3
; 画層のドットペアを入れ替えて、更新します。
❻ (setq ent2 (subst New_lay Old_lay ent2)) ; 新旧の画層名を入れ替える
❼ (entmod ent2) ; 図形情報を更新する（必須）                               → Step4
) ;end
```

番号	関数名	説　明
①	getvar	getvar “CLAYER” で現在画層名が取得できます。 <CLAYER> は現在画層名を格納しているシステム変数です。
②	cons	8 と C_lay の二つをドットペアの組み合わせにします。(8. C_lay)
③	entsel	ユーザーがマウスで選択した図形 (図形名と座標) を変数 ent1 にセットします。 entsel の後に ” と ” に文字を入力すると、コマンドラインに表示されます。 文字を省いた場合は、< 図形を指示 > のメッセージが表示されます。
④	entget	ent1 の中には、図形名と座標値が入っていますが、必要なのは図形名ですから car 関数で第 1 番目の引数 (図形名) を取得しています。
⑤	assoc	図形要素リストから画層の項目だけ取り出します。
⑥	subst	古いドットペア (8. “1”) と、新しいドットペア (8. C_lay) を入れ替えます。 書式は、(subst new old 図形名) です。
⑦	entmod	データを更新します。subst を使用したときは、必ず必要です。

LayChg2.lsp	他の画層にある図形を選択して同じ画層に変更する
目的：	選択した図形の画層を他の画層にある図形を選んで、同じ画層にします。例えば、選択した図形の画層 <1> を他の画層 <0> にある図形を指示して同じ <0> 画層に変更します。
主要関数：	<entsel><car><entget><assoc><subst><entmod>

Step1 ー ① " 画層を変更する図形を指示：" のメッセージを表示して、画層を変更するオブジェクトを選択します。変数 <en1> には下記の情報が取得できます。❶

(< 図形名 : 7ef03990> (1483.01 1390.27 0.0))

② 1 番目のリストが図形名ですから、car 関数で取得します。

(car en1) → < 図形名 : 7ef03990>

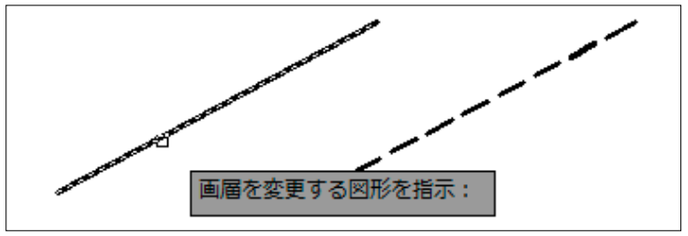
③図形名の中身を展開する関数 entget を使います。❷

(entget (car en1))

((-1 . < 図形名 : 7ef03990>) (0 . "LINE") (330 . < 図形名 : 7ef01cf8>) (5 . "2A2") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 1402.95 1350.28 0.0) (11 1871.01 1556.41 0.0) (210 0.0 0.0 1.0))

④画層名のあるドットペアは、(8 . "0") ですから、これから選択したオブジェクトの画層は "0" であることが判ります。❸

左の線分を選択



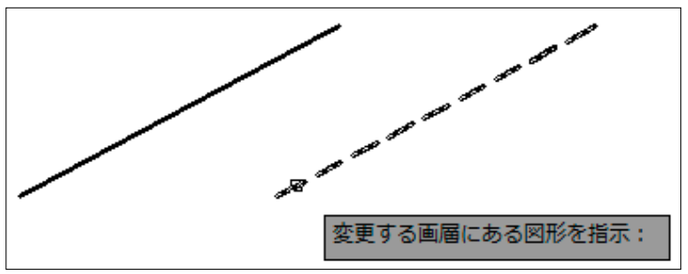
左：画層 <0>、線種 < 実線 >
右：画層 <1>、線種 < 破線 >

Step2 ー ① " 変更する画層にある図形を指示:" のメッセージを出して、変更する画層にあるオブジェクトを選択します。❹

② Step1 と同様にして、このオブジェクトの画層を取得します。❺❻

取得したオブジェクトの画層のドットペアが (8 . "1") とします。

右の線分を選択



左：画層 <0>、線種 < 実線 >
右：画層 <1>、線種 < 破線 >

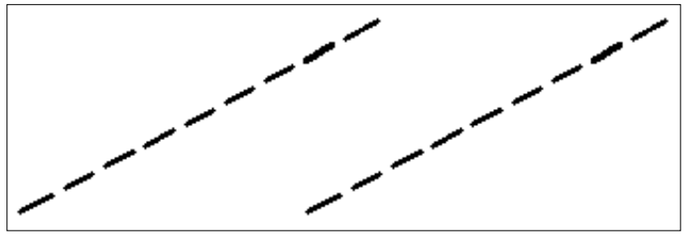
Step3 ー① Step1 で取得した画層のドットペアと Step2 で取得したドットペアを入れ替える関数は subst です。書式は、(subst 新しいドットペア　古いドットペア　図形名) です。❼

(subst lay_new lay_old en1))

② subst 関数で入れ替えた後は、必ず entmod 関数で図形情報を更新します。❽

書式は、(entmod 図形名) です。

左の線分の画層が右の線分の画層と同じになりました

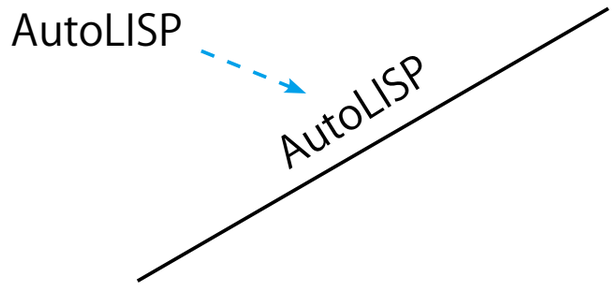


左：画層 <1>、線種 < 破線 >
右：画層 <1>、線種 < 破線 >

```
(defun C:LayChg2(  / en1 lay_old en2 lay_new)
❶ (setq en1 (entsel "¥n 画層を変更する図形を指示："))
❷ (setq en1 (entget (car en1)))  ; 図形の情報を取得
❸ (setq  lay_old (assoc 8 en1))  ; 図形の画層名を取得
                                                  → Step1
; 変更する画層にある図形を指示して、元の図形の画層を変更する
❹ (setq en2 (entsel "¥n 変更する画層にある図形を指示："))
❺ (setq en2 (entget (car en2)))  ; 図形の情報を取得
❻ (setq lay_new (assoc 8 en2))  ; 図形の画層名を取得
                                                  → Step2
; 選択した図形の画層名を入れ替えて、更新する
❼ (setq en1 (subst lay_new lay_old en1))  ; 画層名の入れ替え
❽ (entmod en1)  ; 図形のデータベースを更新
                                                  → Step3
 (princ)
) ;end
```

番号	関数名	説　明
①	entsel	選択したオブジェクトの図形名と座標値のドットペアを取得します。
②	entget	オブジェクトの図形名の中身を展開します。ドットペアのリストが取得できます。
③	assoc	リストから画層のドットペアを取得します。（例）(8 . "0")
④	entsel	選択したオブジェクトの図形名と座標値のドットペアを取得します。
⑤	entget	オブジェクトの図形名の中身を展開します。ドットペアのリストが取得できます。
⑥	assoc	リストから画層のドットペアを取得します。（例）(8 . "1")
⑦	subst	ドットペアを入れ替えます。(subst new old 図形名)
⑧	entmod	subst で入れ替えたオブジェクトを更新します。（必須）

txt_Ro.lsp	選択した文字を指示した線分と同じ傾きにして配置する
目的：	文字を選んで傾いた線分を指示すると、線分と同じ角度に傾きます。その後、文字を自由な位置に移動します。
主要関数：	<entsel><car><entget><assoc><cdr><angle><if><cons><subst><entmod><pause>



Step1 ー ① " 変更する文字を選択：" のメッセージを表示して、変更する文字を選択します。❶

変数 <txt_ob> には下記の情報が取得できます。

(< 図形名 : 7ef07ac0> (919.26 1176.18 0.0))

② 1 番目のリストが図形名ですから、car 関数で取得します。❷

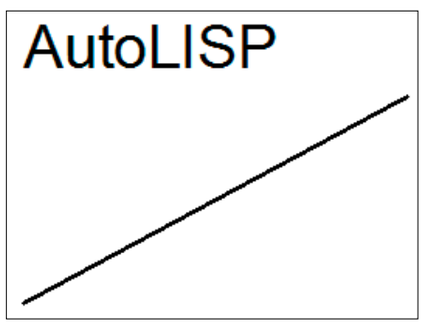
(car txt_ob)) → < 図形名 : 7ef07ac0>

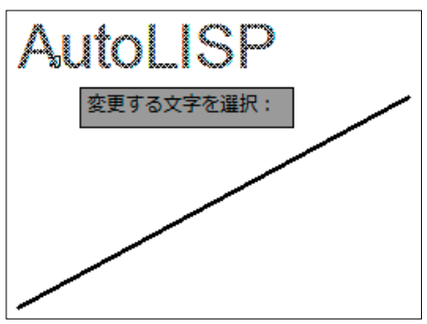
③図形名の中身を展開する entget 関数を使います。❸

(entget txt_ob0)

((-1 . < 図形名 : 7ef07ac0>) (0 . "MTEXT") (330 . < 図形名 : 7ef06cf8>) (5 . "2D8") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbMText") (10 916.12 1178.49 0.0) (40 . 2.5) (41 . 618.96) (46 . 0.0) (71 . 1) (72 . 5) (1 . "AutoLISP") (7 . "STANDARD") (210 0.0 0.0 1.0) (11 1.0 0.0 0.0) (42 . 13.0) (43 . 2.5) (50 . 0.0) (73 . 1) (44 . 1.0))

④角度のあるドットペアは、(50 . 0.0) ですから、これから選択した文字の角度は "0.0" であることが判ります。❹





Step2 ー ① " 位置合わせする線分を選択：" のメッセージを表示して、位置合わせする線分を選択します。❺

変数 <lin_ob> には下記の情報が取得できます。

(< 図形名 : 7ef07af8> (933.975 1169.34 0.0))

② 1 番目のリストが図形名ですから、car 関数で取得します。

(car lin_ob)) → < 図形名 : 7ef07af8>

③図形名の中身を展開する関数 entget を使います。❻

(entget (car lin_ob))

((-1 . < 図形名 : 7ef07af8>) (0 . "LINE") (330 . < 図形名 : 7ef06cf8>) (5 . "2DF") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 924.983 1163.11 0.0) (11 957.785 1185.68 0.0) (210 0.0 0.0 1.0))

④線分には角度のあるドットペアは無いので、線分の始点と終点の座標から角度を計算します。始点のドットペアは (10 924.983 1163.11 0.0) ですから、リストからこの要素を取り出します。❼

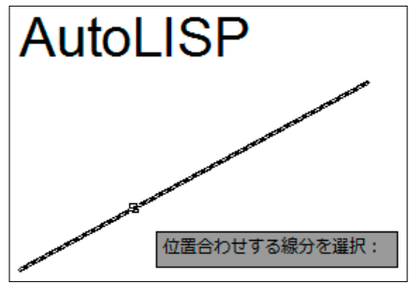
(assoc 10 lin_ob) → (10 924.983 1163.11 0.0)

始点の座標は、ドットペアの 2 番目以降の要素ですから、cdr 関数で取り出します。❽

(cdr (assoc 10 lin_ob)) → (924.983 1163.11 0.0)

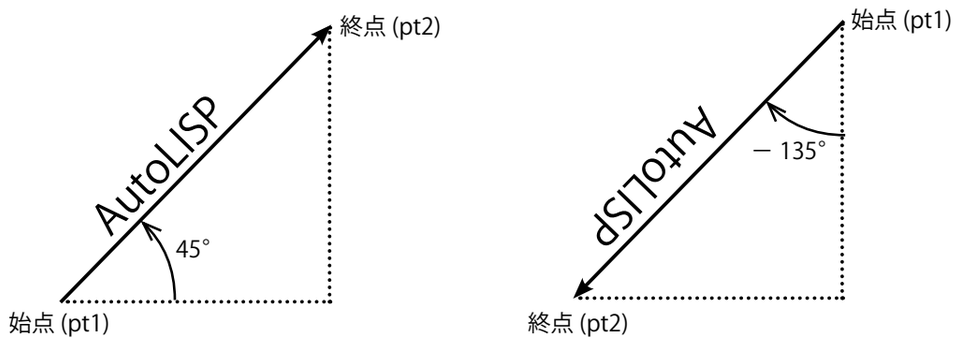
終点の座標も同様にして取り出します。❾❿

(cdr (assoc 11 lin_ob)) → (957.785 1185.68 0.0)



Step3 ー ①文字の角度は 1 つですが、線分の角度は 2 通りあります。

文字の傾きは、<-90° > から <90° > の間ですから、線分の傾きがこの間にあるかどうかを判断し、この傾きでない時は始点と終点の座標を入れ替えます。⓫⓬



まず線分の初期値の傾きを angle 関数で取得します。

(setq ang (angle lin_st lin_en))

この角度 angle（ラジアン）が、90° (1.57 ラジアン) から 270° (4.71 ラジアン) の間であれば、180° (3.14 ラジアン) を差し引きます。（加えても同じです。）

Step4 ー ①文字角度のドットペアと替えるために、線分の角度のドットペアを 作成します。

ドットペアを作成する関数は、cons です。❸

(cons 50 ang)　　→ (50 . ang)

② Step1 で取得した文字角度のドットペアと線分角度のドットペアを入れ替える関数は subst です。書式は、(subst 新しいドットペア　古いドットペア　図形名) ですから、(subst 線分の角度 文字の角度 文字の図形名)) になります。❹

③ subst 関数で入れ替えた後は、必ず entmod 関数で図形情報を更新します。❺

書式は、(entmod 文字の図形名) です。

Step5 ー線分の角度に傾けた文字を、マウスで自由な位置に移動します。❻

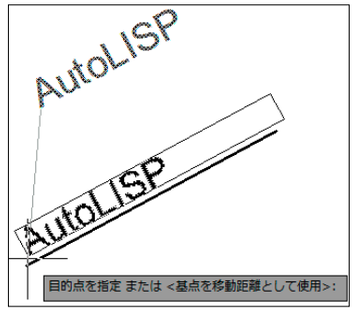
"MOVE" txt_ob0 "" pause pause)

文字オブジェクト <txt_ob0> の次の <""> は選択が終了の意味です。

①最初の pause は移動の基点の指示を待ちます。②次の pause は目的点の指示を待ちます。

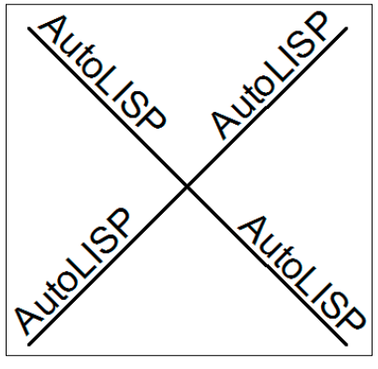
①移動の基点を指示します。

②移動の目的点を指示します。



③回転移動の結果

ISO 基準の文字の回転角度




```
(defun C:txt_Ro(  / txt_ob txt_ob0 txt_ob1 txt_ang_old lin_ob lin_st lin_en ang)
; 角度を変更する文字を選択します。
❶ (setq txt_ob (entsel "¥n 変更する文字を選択 : "))  ; 文字図形を取得
❷ (setq txt_ob0(car txt_ob))  ; 文字の図形名を取得
❸ (setq txt_ob1 (entget txt_ob0))  ; 文字情報（ドット・ペア）のリストを取得
❹ (setq txt_ang_old(assoc 50 txt_ob1))  ; 文字角度のドット・ペアを取得
                                                              → Step1

; 傾いた線分を指示します。
❺ (setq lin_ob (entsel "¥n 位置合わせする線分を選択 : "))  ; 線分を選択
❻ (setq lin_ob (entget (car lin_ob)))  ; 線分情報（ドット・ペア）のリストを取得
❼ (setq lin_st(assoc 10 lin_ob))  ; 線分の始点の座標のドット・ペアを取得
❽ (setq lin_st(cdr lin_st))  ; 始点の座標を変数にセット
❾ (setq lin_en(assoc 11 lin_ob))  ; 線分の終点の座標のドット・ペアを取得
❿ (setq lin_en(cdr lin_en))  ; 終点の座標を変数にセット
                                                              → Step2

; 線分の傾きが、(-90° ) から (+90° ) の間にあるかどうかをチェックします。
⓫ (setq ang (angle lin_st lin_en))  ; 始点から終点への角度を計算
   (if (and (> ang 1.57)(< ang 4.71))  ; もし、角度が 90° から 270° の間だったら
⓬      (setq ang (- ang 3.14))  ; 角度 ang に 180° 減算して、角度を反転する
   ) ;if  ;（または加算）
Point!                                                        → Step3

; 選択した文字の角度を、線分の角度と入れ替えます。
⓭ (setq txt_ang_new (cons 50 ang))  ; 線分角度のドット・ペアを作成
⓮ (setq txt_ob1 (subst txt_ang_new txt_ang_old txt_ob1))  ; 新旧の角度を入れ替える
⓯ (entmod txt_ob1)  ; オブジェクトの更新処理
                                                              → Step4

; 文字を自由な位置に移動します。
⓰ (command "MOVE" txt_ob0 "" pause pause)  ; 回転した文字をマウスで移動
                                                              → Step5
  (princ)
) ;end
```

Point!

ang はラジアンですから、<90° ～ 270° > は <1.57 ～ 4.71 ラジアン > になります。また、<180° > は <3.14 ラジアン > に置き換えて計算します。

番号	関数名	説　明
①	entsel	選択したオブジェクトの図形名と座標値のドットペアを取得します。
②	car	ドットペアの第一番目の要素を取得します。
③	entget	オブジェクトの図形名の中身を展開します。ドットペアのリストが取得できます。
④	assoc	リストから角度のドットペアを取得します。（例）(50 . 角度)
⑤	entsel	選択したオブジェクトの図形名と座標値のドットペアを取得します。
⑥	entget	ドットペアの第一番目の要素を取得し、図形名の中身を展開します。
⑦	assoc	リストから始点のドットペアを取得します。（例）(10 . 始点)
⑧	cdr	ドットペアの第二番目の要素を取得します。（始点の座標）
⑨	assoc	リストから終点のドットペアを取得します。（例）(11 . 終点)
⑩	cdr	ドットペアの第二番目の要素を取得します。（終点の座標）
⑪	angle	2 点間の角度を取得します。
⑫	if	角度が <-90°　> から <+90°　> の間にあるかどうかを判断します。
⑬	cons	新しい角度のドットペアを作成します。（例）(50 . 角度)
⑭	subst	ドットペアを入れ替える関数です。(subst　new　old　図形名)
⑮	entmod	subst で入れ替えたオブジェクトを更新します。（必須）
⑯	"MOVE"	移動コマンドで文字を移動します。（pause はユーザーの指示待ちです。）

Point!

文字を移動する時に、文字の挿入点を変更する（左下から中央下など）ことによって、
選択した文字を線分や円弧の中央に自動的に移動させることができます。

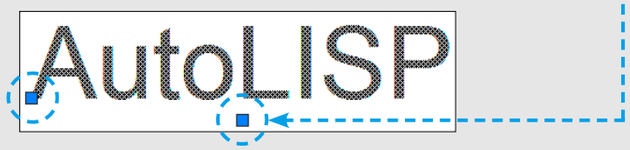
Step1 ー文字を記入した場合、挿入基点の初期値は左下です。

Step2 ーこの文字の図形情報を確認すると、挿入基点は (10 58.185 55.1429 0.0) に表示されます。
(11 0.0 0.0 0.0) は他の挿入基点ですが、無い場合は <0.0 0.0 0.0> になっています。

コマンド : (setq ent1 (entget (entlast)))

((-1 . < 図形名 : 7ef335a0>) (0 . "TEXT") (330 . < 図形名 : 7ef21cf8>) (5 . "22C") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbText") (10 58.185 55.1429 0.0) (40 . 24.713) (1 . "AutoLISP") (50 . 0.0) (41 . 1.0) (51 . 0.0) (7 . "Standard") (71 . 0) (72 . 0) (11 0.0 0.0 0.0) (210 0.0 0.0 1.0) (100 . "AcDbText") (73 . 0))

Step3 ー文字の挿入基点を変更するコマンド [JUSTIFYTEXT] で、新しい挿入点を <BC> に変更します。



Step4 ー再度、この文字の情報を確認します。今度は新しい挿入点の座標 (11 131.093 47.8268 0.0) が
表示されているのが判ります。

コマンド : (setq ent2 (entget (car(entsel))))

((-1 . < 図形名 : 7ef335a0>) (0 . "TEXT") (330 . < 図形名 : 7ef21cf8>) (5 . "22C") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbText") (10 58.185 55.1429 0.0) (40 . 24.713) (1 . "AutoLISP") (50 . 0.0)(41 . 1.0) (51 . 0.0) (7 . "Standard") (71 . 0) (72 . 1) (11 131.093 47.8268 0.0) (210 0.0 0.0 1.0) (100 . "AcDbText") (73 . 1))

第２節　オブジェクトの処理（複数図形）

前節では、オブジェクトを１つずつ編集しましたが、複数のオブジェクトをまとめて編集するには
選択セットというグループにして編集を行います。
AutoLISP には、この選択セットを処理するための多くの関数があります。

選択セットを作成する最も一般的な方法として ssget 関数があります。この関数を使用して、次のいずれかの方法で選択セットを作成できます。

①図面全体を選択する。
②［最後 (L)］、［直前 (P)］、［窓 (W)］、［暗黙 (I)］、［ポリゴン窓 (WP)］、［交差 (C)］、［ポリゴン交差 (CP)］、［フェンス (F)］のいずれかのオプションを使用して、選択するオブジェクトを指定する。
③オブジェクトのプロパティ（画層・色・線種など）で選択するオブジェクトを指定する。
④オブジェクトのジオメトリ（位置・大きさなど）で選択するオブジェクトを指定する。
⑤関係演算子で選択するオブジェクトを指定する。
⑥論理演算子で選択するオブジェクトを指定する。

上記のどの方法を使用しても、フィルタリングを使用して属性や条件のリストを指定し、それらに一致するオブジェクトをまとめて選択できます。

使用頻度の高い AutoLISP の 選択セット操作関数を、次の表に示します。（順不同）

①選択セット操作関数（オブジェクトを選択）	
関　数	説　明
(ssget [mode] [pt1 [pt2]] [pt-list] [filter-list])	オブジェクト (図形) を選択するようユーザーに要求し、選択セットを返します。

②選択セット操作関数（オブジェクトのセットへの追加と削除）	
関　数	説　明
(ssadd [ename [ss]])	選択セットにオブジェクト (図形) を追加、または新しい選択セットを作成します。引数を指定しないと、エンティティを含まない選択セットを作成します。
(ssdel enamess)	選択セットからオブジェクト (図形) を除去します。

③選択セット操作関数（選択セットの情報を取得）	
関　数	説　明
(sslength ss)	選択セットに含まれるオブジェクト (図形) の数を示す整数を返します。
(ssmemb enamess)	指定されたオブジェクト (図形) が選択セットのメンバーかどうかを調べます。含まれている時は、選択セット名を返します。含まれていない時は、nil を返します。
(ssname ss index)	選択セットの指定されたインデックス番号の要素のオブジェクト (図形) 名を返します。番号は、0 から (エンティティ数 -1) の範囲です。

1 選択セット操作関数（オブジェクトを選択）

SSGET	図形群を選択します

(setq en1 (ssget))
オブジェクトを選択 : → メッセージに従って、複数のオブジェクトを選択します。
<Selection set: 4> → コマンドラインに選択セットが表示されます。

コマンド : !en1 → コマンドラインに <!en1> と入力します。
<Selection set: 4> → 選択セットが確認できます。

図形を選択するオプションを以下の表に示します。

選択のオプション	意　味
(ssget "P")	直前に選択した図形群を再度選択します。
(ssget "L")	最後に描いた図形を選択します。
(ssget "W" P1 P2)	窓で囲んだ図形群を選択します。
(ssget "C" P1 P2)	交差で囲んだ図形群を選択します。
(ssget "C" P1 P1)	点 p1 を通る図形群を選択します。
(ssget "WP" P1 P2・・)	多角形窓で囲んだ図形群を選択します。
(ssget "WC" P1 P2・・)	多角形交差で囲んだ図形群を選択します。
(ssget "F" P1 P2・・)	フェンスで囲んだ図形群を選択します
(ssget "X" (list (cons 0 "LINE")))	線分を選択します。

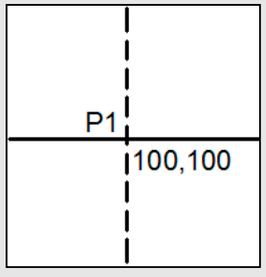
Point!

ssget に対する最初の引数は、どの選択オプションを使用するかを指定する文字列です。
次の 2 つの引数 pt1 と pt2 は、選択オプションに応じて点の値を指定します (省略できます)。
選択オプションとしてポリゴン (フェンス、ポリゴン交差、ポリゴン窓) を使用する場合は、
引数として点のリスト (P1 P2・・) を指定する必要がありますが、最後の引数 (点のリスト) は
省略可能です。

コマンドラインに、(ssget) だけを入力すると、AutoCAD のメッセージが表示されます。
コマンド : (ssget)
オブジェクトを選択 :

Point!

(ssget P1) と (ssget "C" P1 P1) との違いについて



左図のように、点 P1(100,100) で 2 本の線分が交差している場合に、(ssget P1) では、2 本の線分のどちらかしか取得できません。
両方の線分を取得するには、(ssget "C" P1 P1) と記述します。

ssget	複数の図形を選択する

1.entsel 関数は 1 つの図形しか選択できませんが、ssget 関数は窓選択や交差選択を使って、一度に複数の図形を選択することができます。
(ssget)
この関数は < 図形を指示 > のプロンプトを表示して、マウスの右ボタンを押すまで図形の選択状態になります。

2. 例えば、線分と円を選択して変数 ent1 にセットします。(オブジェクトは 2 つ)
(setq ent1 (ssget))

3. この変数 ent1 の中に、オブジェクトがいくつあるかを調べる関数が、sslength です。
(setq ent2 (sslength ent1)) と入力します。
キーボードから <!ent2> と入力すると、コマンドラインに <2> が返ります。

4. 複数のオブジェクトを持つ変数 ent1 から、1 つのオブジェクトを取り出す関数が、ssname です。
(setq ent3 (ssname ent1 0) と入力すると、線分か円のどちらかが変数 ent3 にセットされます。
(番号は、<0> から数えていきます)

5. この次の段階からは先ほどの entsel 関数の場合と同様に、entget 関数を使ってこのオブジェクトの情報を取得していきます。

Point!

複数の図形を選択して、選択した図形がいくつあるかをコマンドラインに表示する処理は次のように記述します。

```
(defun C:T_ssget( / ent1 n_ent)
  (setq ent1 (ssget))                    → < 図形を指示 > のプロンプトが表示される
  (setq n_ent (sslength ent1))           → 選択した図形の数を取得する
  (prompt "¥n 選択した図形の数は ")      → メッセージを表示する
  (princ n_ent)                          → コマンドラインに図形の数を表示する
);end
```

prompt・・・コマンドラインに"" の中の文字を表示します。
princ ・・・後に続く文字または変数の内容をそのままコマンドラインに表示します。

DimUp.lsp	寸法値に直径記号 < Φ > を付加する
目的：	SSGET 関数を使用して、寸法値に直径記号 < Φ > を付加します。 このプログラムは、自動寸法記入された寸法図形にのみ有効です。
主要関数：	<prompt><ssget><dim1>

①コマンドラインに " 直径記号を付加する寸法値を指示：" を表示します。
(prompt "¥n 直径記号を付加する寸法値を指示：")

②寸法オブジェクトを 1 つ選択して、エンターを押します。
(ssget) は複数のオブジェクトを選択できますが、この場合は 1 本だけの処理を行います。

③ (command "DIM1" "NEW" "%%C<>" en1 "")
寸法値に <> が表示されているときは、修正されていない寸法値になります。
もし、"100" などの実際の数字が表示されているときは、寸法値が修正されています。

直径記号（φ）は、<%%C> で表します。

```
(defun C:DimUp( / en1)
; コマンドラインにメッセージを表示
❶ (prompt "¥n 直径記号を付加する寸法値を指示：")
❷ (setq en1 (ssget)) ;1 本の寸法値を選択
❸ (command "DIM1" "NEW" "%%C<>" en1 "") ; 寸法値に < Φ > 記号を付加
❹ (princ) ;<nil> を表示させない
) ;end
```

番号	関数名	説　明
①	prompt	コマンドラインにメッセージを表示します。
②	ssget	オブジェクトの選択を指示する関数です。
③	"DIM1" "NEW" "%%C<>"	寸法値をφ <%%C> に置き換えます。<> は自動寸法の数値を表します。
④	princ	最後の行に、<nil> を表示させません

2 選択セット操作関数（オブジェクトの選択セットへの追加と削除）

SSADD	図形群に図形を新たに加えます

```
(setq  ss1 (ssadd))                → 空の選択セットを作成します。
<Selection set: 6>                 → コマンドラインに空の選択セットが表示されます。
(setq sslen (sslength ss1))        → 選択セット <ss1> にある図形の数を求めます。
0                                  → <0> であることが判ります。

(setq ent1 (car (entsel)))         → 他の図形を選択し、そのエンティティ名を ent1 とします。
オブジェクトを選択 :               → メッセージに従って、1 つのオブジェクトを選択します。

(setq  ss1  (ssadd  ent1  ss1))    → エンティティ <ent1> を選択セット <ss1> に加えます。
(setq sslen (sslength ss1))        → 選択セットの数を求めます。
1                                  → 選択セットの数が確認できます。（1 増加しています）
```

SSDEL	図形群の中から、指定した図形を削除します

```
(setq sslen (sslength ss1))        → 選択セットの数を確認します。
1                                  → 選択セットの数が表示されます。

(setq  ss1  (ssdel  ent1  ss1))    → 選択セット <ss1> の中から、エンティティ <ent1> を削除。
<Selection set: 6>                 → コマンドラインに選択セットが表示されます。
(setq sslen (sslength ss1))        → 選択セット <ss1> にある図形の数を求めます。
0                                  → <0> であることが判ります。（1 減少しています）
```

Point!

ssdel 関数は、図形自体を削除するのではありません。

- 変数 (a) の値を空にする → (setq a nil)
- 図形 (ent1) を図面内から削除する → (command "ERASE" ent1)
- 図形 (ent1) を選択セット (ss1) から削除する → (ssdel ent1 ss1)
- 選択セット (ss1) を空にする → (setq ss1 nil)
- 選択セット (ss1) を削除する → (command "ERASE" ss1)

sslength と ssname の関係を次のプログラムで考えてみましょう。

3 選択セット操作関数（選択セットの情報を取得）

SSNAME	選択した図形群の中から、指定した番目の図形を返します

選択した図形群は、<0> からの連番になります。

(setq en1 (ssget))

オブジェクトを選択 : → メッセージに従って、複数のオブジェクトを選択します。

(setq ss1 (ssname en1 1)) → 選択セットの 2 番目のエンティティを選択します。

コマンド : !ss1 → コマンドラインに < !ss1 > と入力します。

< 図形名 : 7ef03550> → 選択セットの 2 番目のエンティティが表示されます。

SSLENGTH	選択した図形群の数を返します

(setq en1 (ssget))

オブジェクトを選択 : → メッセージに従って、複数のオブジェクトを選択します。

<Selection set: 4> → コマンドラインに選択セットが表示されます。

(setq ss1 (sslength en1)) → 選択した図形の数を求めます。

コマンド : !ss1 → コマンドラインに < !ss1 > と入力します。

2 → 選択した図形の数が確認できます。

SSMEMB	図形群の中に指定した図形があるかどうか調べます

(setq mem1 (ssmemb ss2 ss1)) → エンティティ <ss2> が選択セット <ss1> のメンバーかどうか調べます。

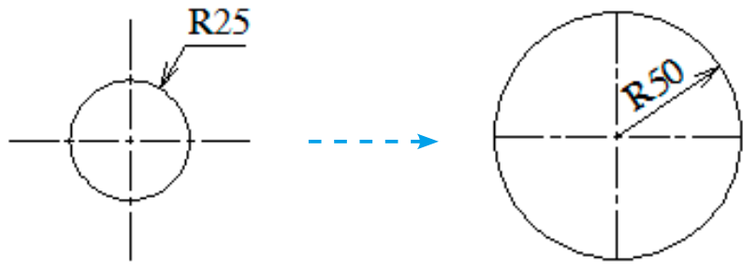
コマンド : !mem1 → コマンドラインに < !mem1 > と入力します。

< 図形名 : 7ef03558> → 図形がある場合は、エンティティ名が表示されます。

nil → 図形が無い場合は、<nil> が表示されます。

sslength と ssname の関係
sslength と ssname の関係を次のプログラムで考えてみましょう。

半径 <25> の円を含む複数の円を ssget 関数で選択し、それぞれの半径を <50> に変更します。

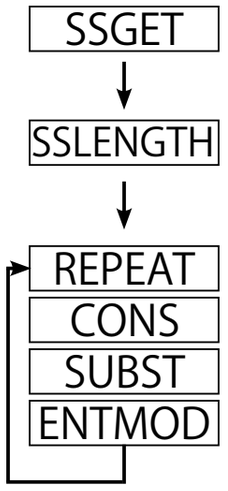


Step1 ー ssget 関数で、複数の円を選択します。
(setq en1 (ssget))

Step2 ー sslength 関数で、選択した図形の数を取得します。
(setq ent_len (sslength en1))

Step3 ー repeat 関数で、以下を ent_len 回繰り返します。（例は半径 <25>）
①半径 <25> と <50> のドット・ペアを作ります。(cons 関数)
②半径 <25> を <50> と入れ替えます。(subst 関数)
③図形情報を更新します。(entmod)

Step4 ー en1 に図形が複数ある場合は、最後の図形まで Step3 を繰り返します。



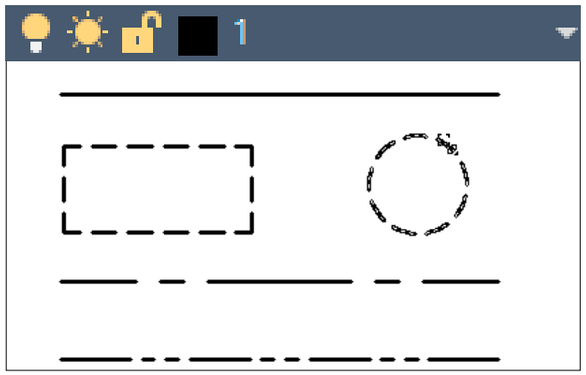
Point!

ssget 関数で取得した図形の情報を 1 つ 1 つ取り出して、変更を加えて行く場合に、一般的に repeat 関数を使用しますが、その repeat する回数は sslength 関数で得ることが出来ます。
ssname 関数は、選択セットから図形を 1 つずつ取り出します。
書式は (ssname 選択セット名 <0 からの番号 >) です。
ですから、sslength 関数と ssname 関数はセットでよく使用されます。

繰り返しの関数には while 関数もありますが、こちらは繰り返しの回数が判らない場合に有効です。
上の例では、選択セットの中が <nil> になるまで繰り返す処理で同じ結果を得ることができます。

del_lay.lsp	選択した図形と同じレイヤーにある図形をすべて削除
目的：	選択した図形と同じレイヤーにある図形をすべて削除します。
主要関数：	<entsel><car><entget><assoc><cdr><cons><list><ERASE>

手順 1 ー画層 <1> の図形 (円) を選択します。



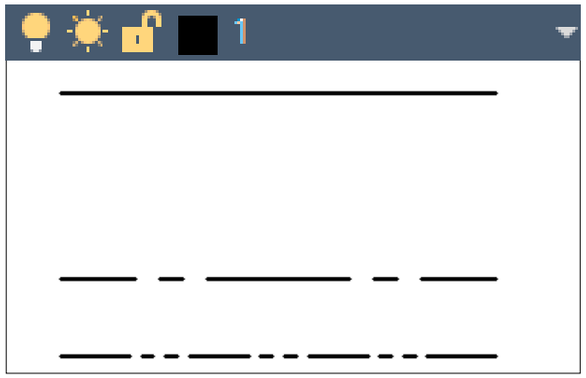
画層 <0>、線種 < 実線 >

画層 <1>、線種 < 破線 >

画層 <2>、線種 < 一点鎖線 >

画層 <3>、線種 < 二点鎖線 >

手順 2 ー同じ画層 <1> の他の図形（長方形）も削除されます。



画層 <0>、線種 < 実線 >

画層 <1>、線種 < 破線 >

画層 <2>、線種 < 一点鎖線 >

画層 <3>、線種 < 二点鎖線 >

Step1 ー削除したいオブジェクトを 1 つ選択します。

① entsel 関数でオブジェクトを選択します。❶

(setq en1 (entsel)) → (< 図形名 : 7ef039a8> (1550.76 814.941 0.0))

Step2 ー① en1 から図形名だけを取り出すには、car 関数を使います。❷

(setq en1 (car en1)) → < 図形名 : 7ef039a8>

②これを個々の要素に展開するのは、entget 関数です。❷

(setq en1 (entget en1))

→ ((-1 . < 図形名 : 7ef03a08>) (0 . "CIRCLE") (330 . < 図形名 : 7ef01cf8>) (5 . "2B1")

(100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "1") (100 . "AcDbCircle")

(10 774.777 1171.83 0.0) (40 . 169.982) (210 0.0 0.0 1.0))

この中にある (8 . "1") が画層の情報です。

③ ssget 関数で、(8 . "1") だけを取り出す関数は、assoc です。❷

(setq ent_lay (assoc 8 en1)) → (8 . "1") が取得できます。

④このドットペアの 2 番目以降を取り出す関数は、cdr です。❷

(setq ent_lay (cdr ent_lay)) → "1" が取得できます。

変数 ent_lay に "1" がセットされます。

Step3 ー① ssget 関数のフィルター用に画層のドットペアを作成します。❸

(list (cons 8 ent_lay)) → cons 関数で (8 . 画層名) のドットペアを作成します。

②選択セット (ssget "X") のフィルタで、この画層のオブジェクトを全て選択します。❸

Step4 ー削除コマンドで、この選択セットを削除します。❹

```
(defun C:del_lay( /  ent_1 ent_lay ss_set)
; オブジェクトを選択します。
❶ (setq en1 (entsel "¥n 削除するレイヤーの図形を指示 :")) ┐→ Step1
; 選択したオブジェクトの画層名を取得します。
❷ (setq ent_lay (cdr (assoc 8 (entget (car en1))))) ┐→ Step2
; 選択したオブジェクトの画層と同じオブジェクトを取得します。
❸ (setq ss_set (ssget "X" (list (cons  8  ent_lay)))) ┐→ Step3
; 選択したセットを削除します。
❹ (command "ERASE" ss_set "") ┐→ Step4
   (princ)
) ;end
```

番号	関数名	説　明
①	entsel	ユーザーが選択した図形を変数 ent1 にセットします。 entsel 関数は 1 つの図形しか選択できません。
②	assoc	画層のグループコードは <8> ですから、entget 関数で取り出した図形情報から画層名を取り出し、その画層名だけを変数 ent_lay にセットします。
③	ssget "X"	図面に存在する図形から、(8 . ent_lay) に合致するオブジェクト取得し、変数 ss_set にセットします。
④	"ERASE"	取得したオブジェクト群を削除します。

M_chglay.lsp	複数の図形を選択して、そのすべての図形の画層を現在層に変更する
目的：	選択した全てのオブジェクトを現在層に移します。
主要関数：	<getvar><cons><ssget><sslength><repeat><ssname><entget><assoc> <subst><entmod>

Step1 ー①現在の画層名を変数 <C_lay> に取得しておきます。❶

(getvar “CLAYER”) → 現在層の <0> が取得されます。

②この画層名のドットペアを作成します。ドットペアを作成する関数は、cons です。❷

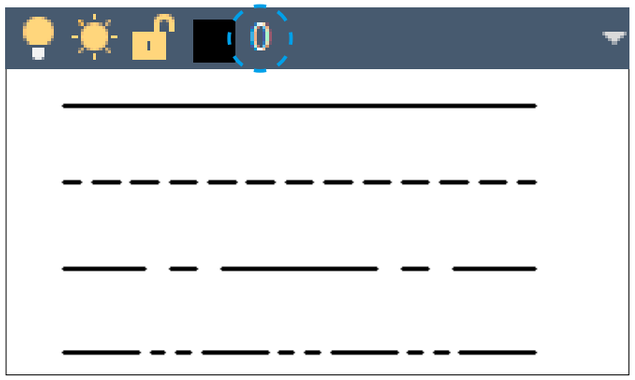
New_lay (cons 8 C_lay) → (8 . "0")

Step2 ー①現在の画層を変更したいオブジェクトを複数選択します。❸

(ssget) 関数は、< オブジェクトを選択 :> のメッセージを表示します。

変数 <ent> には選択セット名 <Selection set: 1d2> が自動的に付けられます。

この例では、現在の画層が <0> の時、画層 <1> と <2> と <3> の図形を選択します。



画層 <0>、線種 < 実線 >

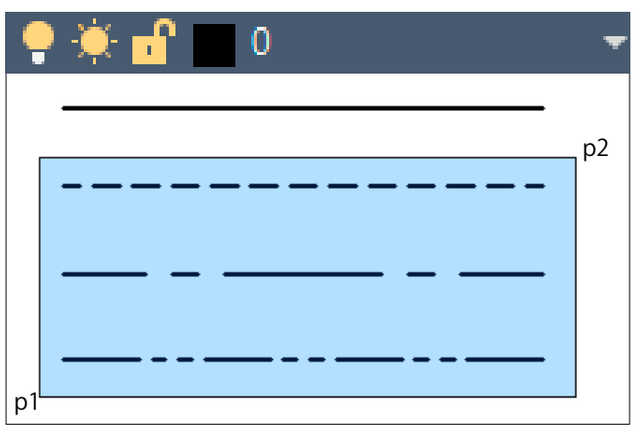
画層 <1>、線種 < 破線 >

画層 <2>、線種 < 一点鎖線 >

画層 <3>、線種 < 二点鎖線 >

②選択したオブジェクトの数を変数 <n_ent> にセットします。❹

(sslength ent) で、選択したオブジェクトの数を取得します。



マウスで p1-p2 と囲って

3 本の線分を選択します。

Step3 ー repeat 関数でオブジェクトの数 (n_ent) だけ、画層の変更を繰り返します。

① ssname 関数でオブジェクトを順番に取り出す <i> の初期値を <0> にセットします。❺

(setq i 0) ← ssname 関数は <0> からカウントします。

②選択セット数 <n_ent> に達するまで繰り返します。❻

③選択セット <ent> から 1 つずつ取り出し、変数 <ent1> にセットます。❼

書式は、(ssname 選択セット名 順番) → ent1 (ssname ent i)

④ entget 関数で ent1 の中身の情報を取得します。(ドット・ペアの集まり) ❽

⑤画層のドット・ペア (8 . 画層名) から、そのオブジェクトの画層のドット・ペアを

変数 <Old_lay> に代入します。❾ → Old_lay (assoc 8 ent2)

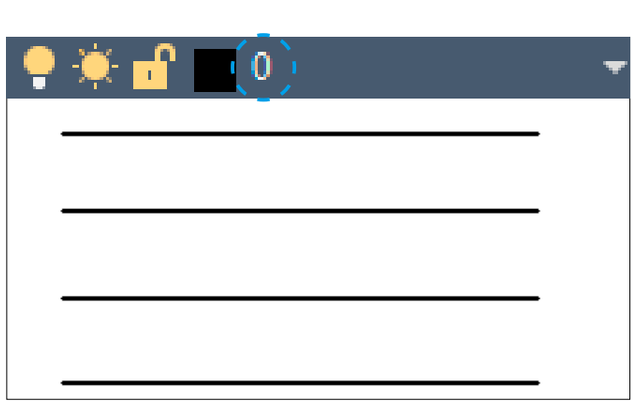
⑥現在層のドット・ペア <New_lay> とオブジェクトの画層のドット・ペア <Old_lay> を

subst 関数で入れ替えます。❿

⑦入れ替えた後は、必ず entmod 関数でオブジェクトを更新します。⓫

⑧選択セットの数 <n_ent> まで、repeat を繰り返します。⓬

選択した図形の画層が < 0 > に変更されました。(画層は変更されません。)



画層 <0>、線種 < 実線 >

画層 <0>、線種 < 実線 >

画層 <0>、線種 < 実線 >

画層 <0>、線種 < 実線 >

```
(defun C:M_chglay( / C_lay New_lay ent1 N_ent i ent2 Old_lay)
;現画層名を取得し、ドットペアを作成します。
❶ (setq C_lay (getvar "CLAYER")) ;現在の画層を取得                    → Step1
❷ (setq New_lay (cons 8 C_lay)) ;画層のドット・ペアを作成
;オブジェクトを選択し、その数をセットします。
❸ (setq ent (ssget)) ;画層を変更する図形を選択                        → Step2
❹ (setq n_ent (sslength ent)) ;選択した図形の数を変数にセット
;オブジェクトの数だけ repeat します。
❺ (setq i 0) ;図形のカウントを <0> にセット（ssname は <0> から始まるため）
❻ (repeat n_ent ;図形の数だけ繰り返す
❼    (setq ent1 (ssname ent i)) ;選択群から 1 つ取り出す
❽    (setq ent2 (entget ent1)) ;その図形のドット・ペアのリストを取得
❾    (setq Old_lay (assoc 8 ent2)) ;画層のドット・ペアを取得          → Step3
❿    (setq ent2 (subst New_lay Old_lay ent2)) ;画層を入れ替える
⓫    (entmod ent2) ;更新処理
⓬    (setq i (1+ i)) ;ssname 関数のカウントを 1 つ加えて、⑥から繰り返す
  ) ;repeat
) ;end
```

番号	関数名	説　明
①	getvar	getvar "CLAYER" で現在の画層名を取得します。
②	cons	画層の要素リスト (ドットペア) を作ります。画層のグループ コードは <8> です。 現在の画層が <1> のときは、変数 C_lay は <1> になっています。 したがって、ここで (8 ． "1") の組み合わせができます。
③	ssget	コマンドラインに < 図形選択：> の文字が表示されます。 ユーザーが選択した複数の図形群が変数 ent にセットされます。
④	sslength	選択した図形群の中に図形がいくつ存在しているかを調べます。 その数を変数 n_ent にセットします。
⑤	(setq i 0)	ssname 関数で取り出すための初期値をセットします。(0 から開始するため)
⑥	repeat	下記⑦から⑪までを、図形の数 (n_ent) だけ繰り返します。
⑦	ssname	図形群から図形を 1 つずつ取り出す関数です。 (ssname　選択セット名　何番目の図形か)
⑧	entget	entget 関数は選択した図形の情報をドットペアのリストで取得します。
⑨	assoc	画層のドットペアを取り出します。 画層のグループ コードは <8> ですから、要素リストが <8> のドットペアを取り出し、変数 Old_lay にセットします。
⑩	subst	要素リスト (ドットペア) を入れ替える関数です。 (subst　新画層名　旧画層名　図形名)
⑪	entmod	データベース情報を更新させる関数です。 最後にこれを宣言しないと、図形は変更されません。
⑫	1+ i	カウント変数 <i> に 1 を加えて、繰り返します。（1 と＋に空白はありません）

Chg_txt.lsp	選択した文字の大きさを指定した大きさに変更する
目的：	選択した全ての文字の大きさを指定した大きさに変更します。
主要関数：	<prompt><ssget><sslength><getdist><cons><repeat><entget><ssname> <assoc><subst><entmod><1+>

Step1 ー① "大きさを変更する文字を指示：" のメッセージを表示して、文字の大きさを変更したい文字を複数選択します。❶

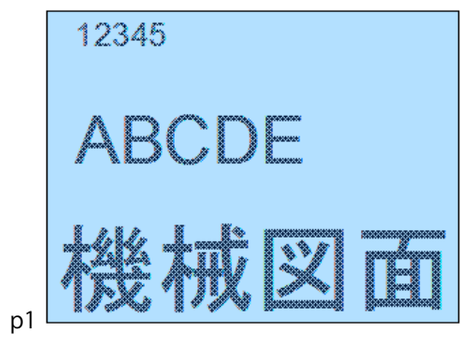
② (ssget) 関数は、< オブジェクトを選択 :> のメッセージを表示します。❷
'((0 . "TEXT")) は選択したオブジェクト群から図形タイプが文字 <TEXT> だけを取得します。

③取得した文字のオブジェクトの数を変数 <ent_len> に代入します。❸

文字高さの違う 3 つの文字を選択し、文字高さを < 5 > に変更します。

Step2 ー① ""新しい文字の大きさを入力：" のメッセージを表示して、新しい文字の大きさを指定します。getdist 関数はキーボードからの入力でもマウスで画面上で長さを指示することもできます。❹

②文字の大きさのドットペア (40 ． 文字の大きさ) 作成し、変数 <new_tsi> に代入します。
→ new_tsi (assoc 40 new_tsi) ❺
これは次の repeat 関数で、1 つ 1 つの文字オブジェクトの文字の大きさのドットペアと入れ替えるために、入れ替え用の新しいドットペアを事前に作成しておくためです。



マウスで p1-p2 と囲って
3 つの文字を選択します。

Step3 ー repeat 関数でオブジェクトの数 (ent_len) だけ、文字の大きさの変更を繰り返します。

① ssname 関数のカウント変数 <i> の初期値を <0> にセットします。❻

(setq i 0)

②選択セットの数 <ent_len> に達するまで繰り返します。❼

③選択セット <ent_obj> から 1 つずつ取り出します。❽

書式は、(ssname 選択セット名 順番) → (ssname ent_obj i)

④ entget 関数で ent_obj の情報を取得します。(ドットペアのリスト)

⑤選択したオブジェクトの文字の大きさのドットペア (40 . 文字の大きさ) を変数 <old_tsi> に代入します。❾ → old_tsi (assoc 40 old_obj)

⑥新しい < 文字の大きさ > のドットペア <new_tsi> と古い < 文字の大きさ > のドットペア <old_tsi> を subst 関数で入れ替えます。❿

書式は、(subst 新しい文字の大きさ 古い文字の大きさ 文字オブジェクト) です。

⑦入れ替えた後は、必ず entmod 関数でオブジェクトを更新します。⓫

⑧選択セットの数 <ent_len> まで、repeat を繰り返します。⓬

選択した文字の高さが <5> に変更されました。

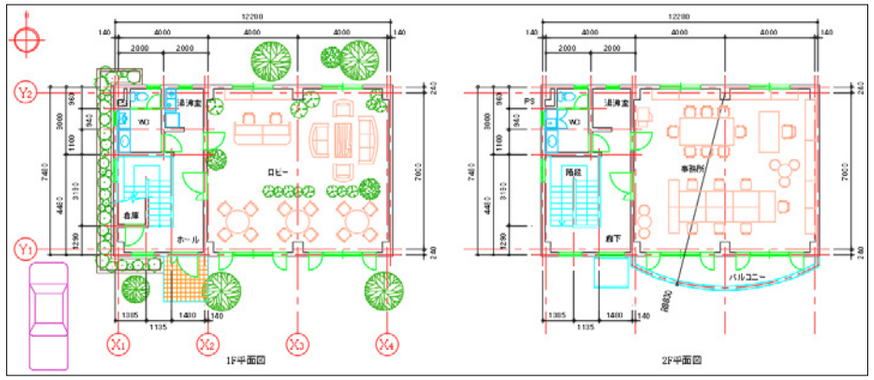

```
(defun C:Chg_txt( / ent_obj ent_len new_tsi i old_obj old_tsi)
;高さを変更する文字を選択して、選択した数を取得します。
❶ (prompt "¥n 大きさを変更する文字を指示 : " )
❷ (setq ent_obj (ssget '((0 . "TEXT" ))))  ;文字オブジェクトだけを取得
❸ (setq ent_len (sslength ent_obj))  ;文字オブジェクトの数を取得
                                                          → Step1
;新しくする文字の大きさを入力して、ドット・ペアを作成する
❹ (setq new_tsi (getdist "¥n 新しい文字の大きさを入力 : " ))
❺ (setq new_tsi (cons 40 new_tsi))  ;新しい文字の大きさのドット・ペアを作成
                                                          → Step2
```



```
❻ (setq i 0)  ;ssname 関数で順番に取り出すカウントの初期値を <0> にセット
❼ (repeat ent_len  ; 選択した文字の数だけ繰り返す
❽   (setq old_obj (entget (ssname ent_obj i)))  ; 選択した文字の 1 つを取得
❾   (setq old_tsi (assoc 40 old_obj))  ; 文字の大きさのドット・ペアを取得
❿   (setq old_obj (subst new_tsi old_tsi old_obj))  ; 文字の大きさを入れ替える
⓫   (entmod old_obj)  ; 図形を更新
⓬   (setq i (1+ i))  ; カウントを <1> 加えて、⑦から繰り返す
  ) ;repeat                                                → Step3
) ;end
```

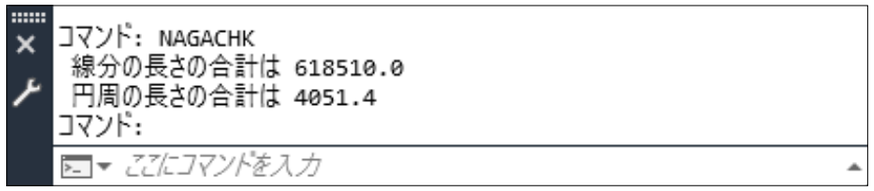
番号	関数名	説 明
①	prompt	" から "の文字をコマンドラインに表示します。
②	ssget	ユーザーに図形を選択させます。 '((0 . "TEXT")) は選択した図形群から文字の図形だけを取得し、変数 ent_obj にセットします。
③	sslength	選択した図形群の中に図形がいくつ存在しているかを調べます。 その数を変数 n_ent にセットします。
④	getdist	文字の高さを入力します。(キーボードからでも図面上からでも構いません。)
⑤	cons	文字の大きさの要素リスト (ドット・ペア) を作成します。 文字高さのグループ コードは <40> ですから、40 の要素リスト (ドット・ペア) を作成し、文字高さの変数 new_tsi をセットします。
⑥	(setq i 0)	ssname 関数で取り出すための初期値をセットします。 (⑧の ssname 関数が <0> から開始するため)
⑦	repeat	下記⑧から⑪までを、文字の数だけ繰り返します。
⑧	ssname entget	図形群から図形を 1 つずつ取り出す関数です。 (ssname 図形群の変数名 何番目の図形) entget 関数は選択した図形の情報を取得します。
⑨	assoc	1 つの文字の要素リスト (ドット・ペア) を取り出します。 文字高さのグループ コードは <40> ですから、40 の要素リスト (ドット・ペア) を取り出し、変数 old_tsi にセットします。
⑩	subst	要素リスト (ドット・ペア) を入れ替える関数です。 (subst 新要素リスト 旧要素リスト 対象の図形名)
⑪	entmod	データベース情報を再計算させる関数です。 最後にこれを宣言しないと、図形は変更されません。
⑫	1+ i	カウント変数 <i> に 1 を加えて、⑦から繰り返します。 <1+> は関数です。(1 と + に空白はありません)

NagaChk.lsp	線分と円の長さの合計を計算する
目的：	図面内にある線分と円の長さの合計を計算します。 P1-80 の（円弧の弧長を測定）と P1-254 の (線分の長さを測定）の応用例です。
主要関数：	<ssget><if><sslength><count><ssname><entget><assoc><cdr><distance> <prompt><princ>



手順 1 ー NagaChk.lsp をロードし、キーボードから <nagachk> と入力します。

手順 2 ー線分の長さの合計と円の長さの合計が、コマンドラインに表示されます。




```
(defun c:NagaChk ( / a1 num nagasa_1 i a2 a3 pt_st pt_en dist b1 nagasa_2 b2 b3 rad)
;線分の長さを加算していく
❶ (setq a1 (ssget "x" '((0 . "LINE") )))  ;図面内の線分のみ取得する
    (if (/= a1 nil)  ;線分の図形が存在すれば、以下の処理を行う
      (progn
❷       (setq num (sslength a1))  ;線分オブジェクトの個数を変数 <num> に代入
❸       (setq nagasa_1 0)  ;長さの初期値を <0> にセット
❹       (setq i 1)  ;個数のカウントを <1> にセット
        (while (<= i num)  ;オブジェクトの数だけ長さを計算する
❺         (setq a2 (ssname a1 (- i 1)))  ;線分を順番に取り出す（一番目は <0> から）
          (setq a3 (entget a2))  ;線分の図形名を変数 <a3> に代入
❻         (setq pt_st (cdr (assoc 10 a3)))  ;線分の始点の座標を変数 <pt_st> に代入
          (setq pt_en (cdr (assoc 11 a3)))  ;線分の終点の座標を変数 <pt_en> に代入
❼         (setq dist (distance pt_st  pt_en))  ;始点と終点の座標から線分の長さを求める
❽         (setq nagasa_1 (+ nagasa_1 dist))  ;長さを加算する
          (setq i (+ 1 i))  ;カウントを 1 つ加えて、次の線分に移る
        );while
      );progn
❾     (setq nagasa_1 0)  ;線分の図形がない場合は、線分の合計長さを <0> にセット
    );if
```



```
;; 円周を加算していく
❿ (setq b1 (ssget "x" '((0 . "CIRCLE") ))) ;図面内の円のみ取得する
    (if (/= b1 nil)  ;円の図形が存在すれば、以下の処理を行う
      (progn
        (setq num (sslength b1))  ;円オブジェクトの個数を変数 <num> に代入
        (setq nagasa_2 0)  ;円周の初期値を <0> にセット
⓫       (setq i 1)  ;個数のカウントを <1> にセット
        (while (<= i num)  ;オブジェクトの数だけ円周を計算する
⓬        (setq b2 (ssname b1 (- i 1)))  ;円を順番に取り出す（一番目は <0> から）
          (setq b3 (entget b2))  ;円の図形名を変数 <b3> に代入
          (setq rad(cdr (assoc 40 b3)))  ;円の半径を取得する
⓭        (setq dist (* 2 pi rad ))  ;円周を計算する
          (setq nagasa_2 (+ nagasa_2 dist))  ;円周を加算する
          (setq i (+ 1 i))  ;カウントを 1 つ加えて、次のオブジェクトに移る
        );while
      );progn
⓮     (setq nagasa_2 0)  ;円の図形がない場合は、円の合計長さを <0> にセット
    );if
;; 合計値をコマンドラインに表示
⓯   (prompt "¥n 線分の長さの合計は ")
    (princ nagasa_1)  ;線分の長さを表示する
⓰   (prompt "¥n 円の長さの合計は ")
    (princ nagasa_2)  ;円周の長さを表示する
(princ)
);end
```

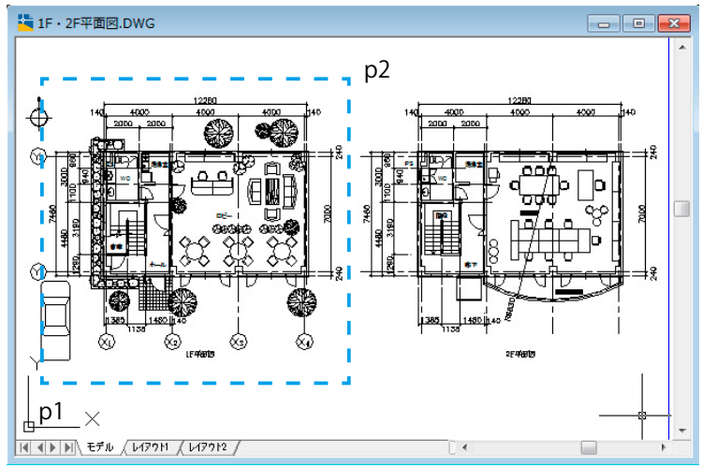
番号	関数名	説　明
①	(ssget "x" '((0 . "LINE")))	図面全体 <(ssget "x")> の中から、線分の図形だけ <'((0 . "LINE"))> を取得して、変数 a1 にセットします。
②	sslength	線分の個数を調べて、変数 num にセットします。
③	nagasa_1	線分の長さの合計を変数 nagasa_1 にセットします。 初期値ですので、最初は <0> をセットします。
④	i	次の ssname 関数のために、カウントの初期値を <1> にセットします。num の数だけ while 以下の処理を行います。
⑤	(- i 1)	ssname は <0> から始まるため、初期値を (- i 1) に設定します。
⑥	pt_st、pt_en	取り出した線分の始点座標と終点座標を変数に代入します。
⑦	distance	始点と終点の座標から、線分の長さを計算します。
⑧	nagasa_1	線分の長さを加算して行きます。
⑨	(setq nagasa_1 0)	この図面に線分の図形がない場合は、線分の長さの合計値に <0> をセットします。
⑩	ssget "x" '((0 . "CIRCLE")))	図面全体 <(ssget "x")> の中から、円の図形だけ <'((0 . "CIRCLE"))> を取得して、変数 b1 にセットします。
⑪	i	次の ssname 関数のために、カウントの初期値を <1> にセットします。num の数だけ while 以下の処理を行います。
⑫	(- i 1)	ssname は <0> から始まるため、初期値を (- i 1) に設定します。
⑬	(* 2 pi rad)	円周を計算します。
⑭	(setq nagasa_2 0)	この図面に円の図形がない場合は、円周の長さの合計値に <0> をセットします。
⑮	(princ nagasa_1)	線分の長さの合計値をコマンドラインに表示します。
⑯	(princ nagasa_2)	円周の長さの合計値をコマンドラインに表示します。

第３節　選択セットのフィルタ

前節で説明したように、図面内にあるオブジェクトをまとめて編集する場合には、ssget 関数を使用すると便利です。
ssget で図面内のオブジェクトを取得する書式は、(setq ent1 (ssget "X")) になります。
これで変数 ent1 には全てのオブジェクトをセットできます。

しかし、選択する図面範囲やオブジェクトの種類が判っていれば、その条件を指定して選択した方が早く選択出来ますし、使用するメモリーも少なくて済みます。
このようにオブジェクトを絞り込んで選択する方法としては、次の４つのパターンがあります。

①選択する範囲をマウスで指定して、その中に含まれる図形を取得する。（範囲選択）
　下図のように、マウスで p1-p2 を囲み、その中に含まれる図形を全て選択します。
　この書式は、(ssget "W" p1 p2) です。



②オブジェクト タイプで選択する。
　例えば、線分だけを選択する場合です。
　この書式は、(ssget "X" ((0 . "LINE"))) です。

③関係演算子で選択する。
　例えば、半径 <10.0> の円だけを選択する場合です。
　この書式は、(ssget "X" '((0 . "CIRCLE") (-4 . "=") (40 . 10.0))) です。

④論理演算子で選択する。
　例えば、半径 10.0 のすべての円と画層 "ABC" 上のすべての線分を選択する場合です。
　この書式は、(ssget "X" '((-4 . "<OR") (-4 . "<AND") (0 . "CIRCLE") (40 . 10.0) (-4 . "AND>") (-4 . <AND")
　　(0 . "LINE")(8. "ABC") (-4 . "AND>") (-4 . "OR>"))) です。

1　範囲を指示して選択する

マウスで指示して選択する範囲を指定するか、選択オプション ("W" や "C"）を使ってオブジェクトを選択します。

点 pt1、pt2、pt3、pt4 を次のようにセットします。
(setq pt1 '(0.0 0.0)　pt2 '(100.0 100.0)　pt3 '(50.0 200.0)　pt4 '(0.0 150.0))
このときのオブジェクトを選択する ssget 関数の使用例は、以下のようになります。

SSGET の使用例	
関数の呼び出し	結　果
(setq ent1 (ssget))	ユーザーに標準の方法でオブジェクトを選択するよう要求し、選択したオブジェクトを選択セットに含めます。
(setq ent1 (ssget "X"))	データベース内のすべてのオブジェクトから選択セットを作成します。
(setq ent1 (ssget "P"))	直前に選択したオブジェクト セットから選択セットを作成します。
(setq ent1 (ssget "L"))	データベースに最後に追加された、画面に表示されているオブジェクトから選択セットを作成します。
(setq ent1 (ssget "C" pt2 pt2))	点 (100,100) を通るオブジェクトから選択セットを作成します。
(setq ent1 (ssget "W" pt1 pt2))	点 (0,0) と点 (100,100) を対角頂点とする窓内のオブジェクトから選択セットを作成します。
(setq ent1 (ssget "F" (list pt2 pt3 pt4)))	点 (100,100)、点 (50,200)、点 (0,150) で定義されるフェンスと交差するオブジェクトから選択セットを作成します。
(setq ent1 (ssget "WP" (list pt1 pt2 pt3)))	点 (0,0)、点 (100,100)、点 (50,200) で定義されるポリゴン内のオブジェクトから選択セットを作成します 。

選択した図形群から、オブジェクトを 1 つずつ取り出して検査するには次のステップを踏みます。
Step1 ーオブジェクト変数 <ent1> に複数の図形を取得します。
　　(setq ent1 (ssget "X"))　　→ 図面内の全てのオブジェクトを選択
　　<Selection set: 820>　　→ 選択セット名が表示されます。
Step2 ー選択セットの中にあるオブジェクト数を取得します。
　　(setq n_ent1 (sslength ent1))　　→ sslength 関数でオブジェクト数を取得
　　680　　→ 前ページの平面図にある図形の数
Step3 ー選択セットの中にあるオブジェクトを 1 つずつ取り出します。
　　(setq obj (ssname ent1 0))　　→ 選択セット <ent1> の 1 番目の図形を取得
　　< 図形名 : 7ef2a1a8>　　→ 1 番目の図形名が表示されます。
Step4 ーこの図形名の中身を entget 関数で表示します。
　　(setq obj_name (entget obj))　　→ entget 関数で図形の内容が判ります。
　　((-1 . < 図形名 : 7ef2a1a8>) (0 . "LINE") (330 . < 図形名 : 7ef13cf8>) (5 . "4B05") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . " 外構 ") (62 . 0) (100 . "AcDbLine") (10 5921.96 21250.1 0.0) (11 7921.96 21250.1 0.0) (210 0.0 0.0 1.0))
Step5 ーこの中にあるリストの各項目（ドット・ペア）を変更します。
　　繰り返し処理関数が使われます。（repeat 関数で <n_ent1> の数だけ繰り返します。）

2 オブジェクト タイプで選択する

ssget 関数は、図形名 (グループ -1)、ハンドル (グループ 5)、および拡張データ (Xdata) コード (1000 以上のグループ) 以外のすべてのグループ コードを認識しますが、無効なグループ コードを指定した場合は、ssget により無視されます。

次の表に、各種のオブジェクト選択オプションでフィルタ リスト を使用する例を示します。

フィルタ リストを使用した SSGET の例	
関数の呼び出し	結　果
(setq ent1 (ssget '((0 . "TEXT"))))	ユーザーに標準の方法でオブジェクトを選択するよう要求し、文字オブジェクトだけを選択セットに追加します。
(setq ent1 (ssget "X"'((0 . "CIRCLE"))))	データベース内のすべての円オブジェクトの選択セットを作成します。
(setq ent1 (ssget "P" '((0 . LINE"))))	直前に作成された選択セットからすべての線分オブジェクトを含む選択セットを作成します。
(setq ent1 (ssget "W" pt1 pt2 '((8 . "FLOOR"))))	画層 FLOOR 上にある、窓内のすべてのオブジェクトの選択セットを作成します。
(ssget "I" '((0 . "LINE") (62 . 5)))	暗黙の選択セット (PICKFIRST が有効なときに選択されたオブジェクト セット) に含まれるすべての青の線分オブジェクトの選択セットを作成します。 このフィルタによって色 5(青) が割り当てられている線分が選択されますが、ByLayer または ByBlock プロパティによって適用されている色を持つ青の線分は選択されません。

グループコードと必要な値が分かっている場合は、上記の例のように、リストにクォーテーションを使用できます < '((0 . "LINE") (62 . 5))>。
どちらか一方を変数で指定する場合は、list と cons 関数を使用してリストを作成します。

①次のコードは、画層 FLOOR 上にある、図面内の全てのオブジェクトの選択セットを作成します。
(setq lay_name "FLOOR")
(setq ent1 (ssget "X" (list (cons 8 lay_name))))

②また、次の例のようにフィルタ リスト に複数の特性を指定すると、全ての指定条件を満たす図形だけが選択セットに含まれます。
(ssget "X" (list (cons 0 "CIRCLE")(cons 8 lay_name)(cons 62 1)))
このコードは、画層 FLOOR 上にある赤色の円オブジェクトだけを選択します。

③また、次の例のように <"X"> を記述せずにフィルタ リスト に複数の特性を指定すると、前ページと同様にマウスで指定した範囲内で条件を満たす図形だけが選択セットに含まれます。
(ssget (list (cons 0 "CIRCLE")(cons 8 lay_name)(cons 62 1)))
このコードは、マウスで指定した範囲内で、画層 FLOOR 上にある赤色の円オブジェクトだけを選択します。

3 関係演算子で選択する

図面内のオブジェクトの選択に際して、最初から選択するオブジェクトを絞ることができます。
(ssget "X") の後に、関係演算子を指定する特殊なグループ コード -4 を含めることで選択を指定できます。
書式は、(ssget "X" '((0 . 図形名) (-4 . " 関係演算子 ") (グループコード . 条件内容))) です。

次の表に、指定可能な関係演算子を示します。

選択セット フィルタ リストの関係演算子	
演算子	説　明
"*"	すべてのもの (常に真)
"="	等しい
"!="	等しくない
"/="	等しくない
"<>"	等しくない
"<"	より小さい
"<="	より小さいか等しい
">"	より大きい
">="	より大きいか等しい
"&"	ビット方式の AND(整数グループのみ)
"&="	ビット方式のマスクが等しい (整数グループのみ)

関係演算子の使用法は、リストするグループの種類によって異なります。

①ビット演算子 ("&" と "&=") を除くすべての関係演算子は、実数値グループと整数値グループの両方に対して有効です。
半径 (グループ コード 40) が <20.0> 以上 のすべての円を選択するには、
→ (ssget "X" '((0 . "CIRCLE") (-4 . ">=") (40 . 20.0))) になります。

②ビット演算子 "&" と "&=" は、整数値グループに対してだけ有効です。
直径寸法と半径寸法以外の寸法を選択するには、
→ (ssget "X" '((0 . "DIMENSION")(-4 . "<NOT")(-4 . "<OR")(-4 . "&=")(70 . 3)(-4 . "&=")(70 . 4)(-4 .R>")
(-4 . "NOT>"))) になります。

③点グループの場合、X、Y、Z に対するリストを結合して単一の文字列にできます。
円の中心の Y 座標が <100> より上にある円を選択するには、
→ (ssget "X" (list(cons 0 "CIRCLE")(cons -4 "*,>,*")(cons 10 '(0.0 100.0 0.0)))) になります。

④文字列グループには関係演算子を使用できません。ワイルドカードでリストします。
マルチテキストとダイナミック文字の両方を選択するには、
→ (ssget "X" '((0 . "*TEXT"))) になります。

4 論理演算子で選択する

次の表に示す論理グループ演算子を使用してリストしたブール式を作成することにより、グループをリストすることもできます。

選択セット フィルタ リストのグループ演算子		
開始演算子	囲まれるもの	終了演算子
"<AND"	1 つ以上のオペランド（演算の対象）	"AND>"
"<OR"	1 つ以上のオペランド（演算の対象）	"OR>"
"<XOR"	2 つのオペランド（演算の対象）	"XOR>"
"<NOT"	1 つのオペランド（演算の対象）	"NOT>"

グループ演算子は、関係演算子と同じように、グループ コード -4 で指定します。グループ演算子はペアで使用するため、フィルタ リスト内で正しく対応されていなくてはなりません。
対応が正しくないと ssget の呼び出しは失敗します。

① "<AND" （1 つ以上のオペランド） "AND>" →すべての条件にマッチする図形を選択
次の式は、画層 < 中心線 > にある線分を取得します。
(setq ent1 (ssget "X" '((-4 . "<and") (0 . "LINE") (8 . " 中心線 ") (-4 . "and>"))))
(setq ent_len (sslength ent1)) → ent_len の値は、<40> が返ります。

② "<OR" （1 つ以上のオペランド） "OR>" →いずれかの条件にマッチする図形を選択
次の式は、画層 < 中心線 > にあるオブジェクトとすべての画層にある線分を取得します。
(setq ent1 (ssget "X" '((-4 . "<or") (0 . "LINE") (8 . " 中心線 ") (-4 . "or>"))))
(setq ent_len (sslength ent1)) → ent_len の値は、<456> が返ります。

③ "<XOR" （2 つのオペランド） "XOR>" →どちらかの条件にマッチする図形を選択
次の式は、画層 <0> にある線分か画層 < 中心線 > にある線分を取得します。
(setq ent1 (ssget "X" '((0 . "LINE") (-4 . "<XOR")(8 . "0")(8 . " 中心線 ") (-4 . "XOR>"))))
(setq ent_len (sslength ent1)) → ent_len の値は、<45> が返ります。

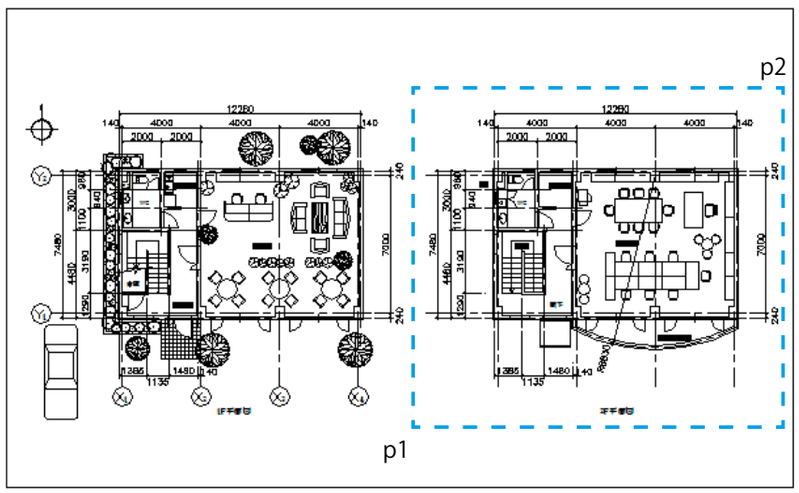
④ "<NOT" （1 つのオペランド） "NOT>" →この条件以外の図形を選択
次の式は、線分以外の図形を取得します。
(setq ent1 (ssget "X" '((-4 . "<NOT") (0 . "LINE") (-4 . "NOT>"))))
(setq ent_len (sslength ent1)) → ent_len の値は、<191> が返ります。

Point!

②の "<OR" (A) (B) "OR>" と③の "<XOR" (A) (B) "XOR>" は同じように見えますが、
②は、条件 (A) か条件 (B)、および両方の条件を満たしている図形を選択します。
一方③は、条件 (A) か条件 (B) のどちらかを満たす図形だけを選択します。

フィルターでリストを作成する

下図を使って、フィルターで選択します。



①図面内の全てのオブジェクトを選択する。
(setq ent1 (ssget "X"))
(setq ent_len (sslength ent1)) → ent_len の値は、<604> が返ります。

② p1 ～ p2 の四角形の中にあるオブジェクトを選択する。
(setq ent1 (ssget)) →マウスで p1 から p2 を囲みます。
(setq ent_len (sslength ent1)) → ent_len の値は、<242> が返ります。

③図面内のマルチテキストとダイナミック文字の両方を選択する。
(setq ent1 (ssget "X" '((0 . "TEXT,MTEXT"))))
(setq ent_len (sslength ent1)) → ent_len の値は、<26> が返ります。

④画層 <0> のオブジェクトを選択する。
(setq ent1 (ssget "X" '((8 . "0"))))
(setq ent_len (sslength ent1)) → ent_len の値は、<18> が返ります。

⑤画層 < 記号 > にある円を選択する。
(setq ent1 (ssget "X" (list (cons 0 "CIRCLE")(cons 8 " 記号 "))))
(setq ent_len (sslength ent1)) → ent_len の値は、<31> が返ります。

⑥画層 < 記号 > にある半径 100 ミリ以下の円を選択する。
(setq ent1 (ssget "X" '((0 . "CIRCLE")(cons 8 " 記号 ")(-4 . "<=")(40 . 100.0))))
(setq ent_len (sslength ent1)) → ent_len の値は、<11> が返ります。

CircleErase.lsp	選択セットの条件を絞り込んで円を一括削除
目的：	図面内にある半径 5 ミリ以下の円を削除します。
主要関数：	<getreal><cons><list><ssget><sslength><prompt><princ><ERASE>

手順 1 ー CircleErase.lsp をロードし、キーボードから <CircleErase> と入力します。

手順 2 ー「何ミリ以下の円を削除しますか？」と聞いてきますので、キーボードから <5> と入力します。

手順 3 ー 5 ミリ以下の円が削除され、コマンドラインに削除された円の数が表示されます。

Point!

②の [5 ミリ以下の円] は、(list(cons 0 "CIRCLE")(cons -4 "<=")(cons 40 5.0)) です。

もし、[3 ミリ以下、7 ミリ以上の円] とするには、次のように記述します。

(ssget "X"(list (cons 0 "CIRCLE")(cons -4 "<or")(cons -4 "<=")(cons 40 3.0)(cons -4 ">=")(cons 40 7.0)(cons -4 "or>")))

結果、下図のように半径 3 ミリ以下、7 ミリ以上の円が削除されます。

Step1 ー何ミリ以下の円を削除するかを入力します。(例は 5 ミリ) ❶

Step2 ー 5 ミリ以下の円を取得するため、list 関数で 1 つのリストにします。❷
ドットペアは cons 関数で作ります。
図形は円の組み合わせ　→ (cons 0 "CIRCLE")　→ (0 . "CIRCLE")
何ミリ以下の組み合わせ → (cons -4 "<=")　→ (-4 . "<=")
半径が 5.0 の組み合わせ → (cons 40 5.0)　→ (40 . 5.0)

Step3 ー選択した円の数を変数 <ent_num> にセットします。❸
(最後にコマンドラインに表示するためです。)

Step4 ー選択した円のセットを削除します。❹

Step5 ーコマンドラインに削除した数を表示します。❺

```
(defun C:CircleErase ( / rad1 ss_set ent_num)
❶ (setq rad1 (getreal "¥n 何ミリ以下の円を削除しますか？ ")) ; 円の半径を入力する          → Step1
; 指定した半径に合致した円を取得します。
❷ (setq ss_set (ssget "X" (list(cons 0 "CIRCLE")(cons -4 "<=")(cons 40 5.0)))) → Step2
;ss_set の数を取得しておきます。(削除した数を最後に表示するためです。)
❸ (setq ent_num (sslength ss_set)) ; 選択セット (ss_set) の数を取得する          → Step3
; 選択したセットを削除します。
❹ (command "ERASE" ss_set "") ; 選択セット (ss_set) を削除する                  → Step4
; コマンドラインに削除した円の数を表示します。
❺ (prompt "¥n 削除した円の数は：") ; メッセージをコマンドラインに表示する
  (princ ent_num)                ; 削除した円の数を表示する                     → Step5
  (princ)
);end
```

番号	関数名	説　明
①	getreal	削除する円の半径を入力します。
②	ssget "X" (list	(list・・以下の条件を絞り込んで、条件に合致した円を取得します。
③	sslength	選択セットに含まれるオブジェクトの数を取得します。
④	"ERASE"	削除コマンドです。最後の <""> は選択の終了を意味します。
⑤	(princ ent_num)	コマンドラインに削除した数 (ent_num) を表示します。

LineErase.lsp	選択セットの条件を絞り込んで線分を一括削除
目的：	図面内にある指定した長さの線分を削除します。 スキャナーで取り込んだラスターデータをベクトル変換したときに、短い線分が生じた時に一度に削除できます。
主要関数：	<getreal><ssget><sslength><ssname><entget><assoc><distance>

線分は長さ情報を持たないので、始点と終点の座標から長さを計算します。

```
(defun C:LineErase(  / lin1 ss_set ent_num i ent1 ent2 ps pe dis1)
❶ (setq lin1 (getreal "¥n 何ミリ以下の線分を削除しますか？ "))  ; 削除する線分の長さを指定する
; 図面内の全線分を取得します。
❷ (setq ss_set (ssget "X" '((0 . "LINE"))))  ; 図面にある全線分を取得する
;ss_set の数を取得しておきます。
❸ (setq ent_num (sslength ss_set))  ; 取得した線分の数をセットする (repeat 関数で使用 )
❹ (setq i 0)  ;repeat の初期値を <0> にセットする (ssname 関数は、<0> から始まるため )
❺ (repeat  ent_num  ; 取得した線分の数だけ、以下を繰り返す
❻   (setq  ent1 (ssname  ss_set  i))  ; 取得した選択セットから、順番に線分を取り出す
❼   (setq  ent2 (entget  ent1))  ; 線分の全ての図形情報を取り出す
❽   (setq  ps (cdr  (assoc  10 ent2)))  ; 始点の座標を取得する
❾   (setq  pe (cdr  (assoc  11 ent2)))  ; 終点の座標を取得する
❿   (setq  dis1  (distance  ps  pe))  ; 始点座標と終点座標から、線分の長さを計算する
⓫   (if  (<=  dis1 lin1)(command "ERASE"  ent1 ""))  ; 指定した長さ (lin1) より短ければ、削除する
⓬   (setq i (1+ i))  ; カウントを 1 増やして⑤に戻る
  );repeat
(princ)
);end
```

番号	関数名	説　明
①	getreal	削除したい線分の長さを指定します。
②	(ssget "X" '((0 . "LINE")))	図面内の線分を取得します。(0 . "LINE") で線分を取得します。
③	(sslength ss_set)	取得した線分の数を取得します。
④	(setq i 0)	ssname 関数で取り出すための初期値をセットします。(0 から開始)
⑤	repeat	ent_num の数だけ繰り返します。
⑥	(ssname ss_set i)	線分の選択セット (ss_set) から順番に取り出します。
⑦	(entget ent1)	取り出した線分の図形情報を取得します。
⑧	(cdr (assoc 10 ent2))	線分の始点のドット・ペア (10 . 始点座標) から始点座標を取り出します。
⑨	(cdr (assoc 11 ent2))	線分の終点のドット・ペア (11 . 終点座標) から終点座標を取り出します。
⑩	(distance ps pe)	始点座標と終点座標から、線分の長さを計算します。
⑪	(<= dis1 lin1)	その長さ (dis1) が指定した長さ (lin1) より短ければ、削除します。
⑫	(setq i (1+ i))	カウントを 1 プラスして、⑤に戻り繰り返します。

第４節　拡張データ

拡張データは、線分や円などの図形に付加される文字情報です。
似た機能に属性定義がありますが、属性定義がブロック図形に対してのみ付加できるのに対して、XDATA は図形オブジェクトと非グラフィカル オブジェクトの両方に持たせることができます。
図形オブジェクトに文字情報を付加することによって、Excel や Access などの外部プログラムに接続して、データベースとして管理することが可能です。

既定では、AutoCAD のハッチング パターンには拡張データが含まれています。下図のハッチングで確認します。

①次のコードはハッチングの XDATA を取得します。

コマンド : (entget (car (entsel)) '("ACAD"))
オブジェクトを選択 : ((-1 . < 図形名 : 7ef039f8>) (0 . "HATCH") (330 . < 図形名 : 7ef01cf8>) (5 ."2AF") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 ."AcDbHatch") (10 0.0 0.0 0.0) (210 0.0 0.0 1.0) (2 . "ANSI37") (70 . 0) (71 .1) (91 . 1) (92 . 7) (72 . 0) (73 . 1) (93 . 4) (10 1779.2 1923.62 0.0) (10 1138.76 1923.62 0.0) (10 1138.76 1647.97 0.0) (10 1779.2 1647.97 0.0) (97 . 1) (330 . < 図形名 : 7ef039a0>) (75 . 1) (76 . 1) (52 . 0.0) (41 . 30.0) (77 . 0) (78 .2) (53 . 0.785398) (43 . 0.0) (44 . 0.0) (45 . -67.3519) (46 . 67.3519) (79 . 0) (53 . 2.35619) (43 . 0.0) (44 . 0.0) (45 . -67.3519) (46 . -67.3519) (79 . 0) (47 . 4.92221) (98 . 1) (10 1350.59 1761.18 0.0) (-3 ("ACAD" (1010 0.0 0.0 0.0))))

②上記の情報の中で、最後の <-3> 以降がこのハッチングに付加されている XDATA です。

使用頻度の高い AutoLISP の 拡張データ処理関数と関連する関数の概要を、次の表に示します。

①拡張データ処理関数	
関　数	説　明
(regapp application)	拡張オブジェクト データを使用できるようにするために、現在の AutoCAD の図面にアプリケーション名を登録します。

②リストに要素を加える関数	
関　数	説　明
(append lst ...)	任意の数のリストを受け取り、それらを 1 つのリストにまとめます。

③オブジェクト処理関数 (オブジェクトの更新)	
関　数	説　明
(entmod 特定リスト)	オブジェクトの定義データを更新します。

拡張データのグループ コードと意味		
図形名	グループ コード	意　味
文字列	1000	拡張データ内の文字列は、最大 255 バイトの長さです (256 番目のバイトは null 文字用に確保されています)。
アプリケーション名	1001 これも文字列値	アプリケーション名は、最大 31 バイトです (32 バイト目は null 文字のために予約されています)。 注:ユーザ独自の拡張データにグループ 1001 を追加しないでください。追加すると、AutoCAD はそれを、新しいアプリケーション拡張データの開始とみなします。
制御文字列	1002	拡張データの制御文字列は、"{" か "}" のどちらかが可能です。これらの括弧により、アプリケーションは、そのデータをリストに細分化して組織化できるようになります。 左側の中括弧はリストの始まりで、右側の中括弧はリストを終了します。リストはネストできます。AutoCAD は、個々のアプリケーションに対する拡張データを読み込むとき、括弧の数が合っているかどうかを調べます。
画層名	1003	拡張データに関連付けられた画層の名前
バイナリ データ	1004	バイナリ データは、可変長の「チャンク」に組織化されます。各チャンクは、最大 127 バイトです。ASCII DXF ファイルでは、バイナリ データは、バイナリ バイト当り 2 文字の 16 進数文字列で表されます。
データベース ハンドル	1005	図面データ内の図形のハンドル 注 : ハンドルおよび拡張データ ハンドルを有する図面が、INSERT[ブロック挿入] コマンド、INSERT *(挿入時にブロック内のオブジェクトを分解) コマンド、XREF[外部参照]/[バインド (B)] コマンド、XBIND[個別バインド] コマンド、または OPEN[開く] コマンドの [部分的に開く] オプションを使用して他の図面に読み込まれるとき、拡張データ ハンドルは、一致する図形ハンドルと同じ方法で変換されるので、これらの結合は維持されます。これは、ブロックに対する EXPLODE[分解] コマンドや他の AutoCAD の操作に対しても行われます。AUDIT[監査] コマンドにより、図面内の図形のハンドルと一致しない拡張データ ハンドルが検出されると、エラーとみなされます。AUDIT が図形を修復すると、ハンドルは 0 に設定されます。
3 つの実数	1010, 1020, 1030	3 つの実数、X、Y、Z の順。これらは点またはベクトル レコードとして使用できます。AutoCAD が値を変更することは決してありません。
ワールド空間の位置	1011, 1021, 1031	単純な 3D 点とは違い、ワールド空間座標は、拡張データが属する親図形とともに移動され、尺度変更され、鏡像化されます。ワールド空間位置は、STRETCH[ストレッチ] コマンドが親図形に適用されこの点が選択窓内にある場合も、ストレッチされます。
ワールド空間の変位	1012, 1022, 1032	これも、親図形とともに尺度変更、回転、鏡像化される 3D 点 (移動やストレッチはされません)
ワールド方向	1013, 1023, 1033	これも、親図形とともに回転および鏡像化される 3D 点 (移動、尺度変更、ストレッチはされません)
実数	1040	実数値
距離	1041	親図形とともに尺度変更される実数値
尺度係数	1042	親図形とともに尺度変更される実数値。距離と尺度係数の違いは、アプリケーション定義です。
整数	1070	16 ビット整数 (符号付きまたは符号なし)
長整数	1071	32 ビット符号付き (長) 整数

1 アプリケーション名を登録する

アプリケーション名は付加される拡張エンティティデータだけでなく、APPID テーブルも同時に登録します。
AutoLISP では regapp 関数によって登録できます。regapp 関数は使用するアプリケーションの文字列を指定する関数です。

```
アプリケーション名 [cost] の登録処理

(setq appname "cost" )  ;アプリケーション名を決めます。
(if (tblsearch "appid" appname)  ;同じ名前が有るかどうかを調べます。
   (prompt (strcat appname "は登録済みです。" ))  ;同じ名前があれば、メッセージを表示して終了します。
   (regapp "cost" )  ;無ければ、アプリケーション名として登録します。
)
```

番号	関数名	説　明
①	setq	アプリケーション名の変数 [appname] に cost を割り当てます。
②	tblsearch	この図面中の APPID テーブルに cost が割り当てられているかどうか調べます。
③	promt	cost が割り当てられていれば、そのメッセージを表示して終了します。
④	regapp	割り当てられていなければ、アプリケーション名として登録します。

Point!

上記のプログラムで登録したアプリケーション名を確認します。

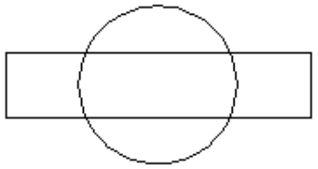
① tblsearch 関数で、アプリケーション名 [cost] を検索します。
コマンド : (tblsearch "appid" appname)

②以下のメッセージが表示されます。
((0 . "APPID") (2 . "cost") (70 . 0))

AutoCAD エンティティのグループ コード	
グループ コード	意　味
0	エンティティ タイプ（"APPID" はアプリケーション名であることを示す）
2	アプリケーションの名前
70	フラグ（0 は無し ）

2 図形に拡張データ (XDATA) を付加する

次のブロック図形に拡張データ (XDATA) を付加します。
付加するデータは、製品名（< 電灯 > 文字列)、制作年（<2020> 整数)、価格（<100.0> 実数）です。



①拡張データを付加する図形 (上のブロック図形）を選択します。(ブロック名は "cost")

```
(setq ent1 (entget (car (entsel))))
→ ((-1 . < 図形名 : 7ef03cb8>) (0 . "INSERT") (330 . < 図形名 : 7ef01cf8>) (5 . "2FF") (100 . cDbEntity")
   (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbBlockReference") (2 . "cost") (10 1318.58 1420.44 0.0)
   (41 . 1.0) (42 . 1.0) (43 . 1.0) (50 . 0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0) (210 0.0 0.0 1.0))
```

②アプリケーション名を設定します。

```
(regapp "light")
```

→ "light" がコマンドラインに表示されます。

③文字列（電灯)、整数値（2020)、実数値（100.0）を付加します。

```
(setq xdata1 '((-3 ("light" (1000 . " 電灯 ")(1070 . 2020)(1040 . 100.0)))))
```

→ ((-3 ("light" (1000 . " 電灯 ") (1070 . 2020) (1040 . 100.0))) がコマンドラインに表示されます。

④選択したブロック図形のリストに追加します。

```
(setq ent1   (append ent1 xdata1))
```

→ ((-1 . < 図形名 : 7ef03cb8>) (0 . "INSERT") (330 . < 図形名 : 7ef01cf8>) (5 . "2FF") (100."AcDbEntity")
(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbBlockReference")(2 . "cost")
(10 1318.58 1420.44 0.0) (41 . 1.0) (42 . 1.0) (43 . 1.0) (50 .0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0)
(210 0.0 0.0 1.0) (-3 ("light" (1000 . " 電灯 ") (1070 . 2020) (1040 . 100.0))))

⑤拡張データを追加したブロック図形のデータベースを更新します。（必須）

```
(entmod ent1)
```

→④と同じ情報がコマンドラインに表示されます。

⑥新しい拡張データ が図形にアタッチされたことを確認するには、次のコードを入力し図形を選択します。

```
(entget (car (entsel)) '("light"))
```

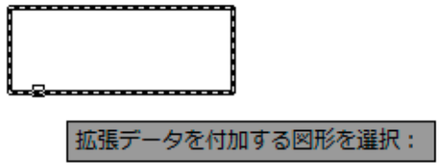
→④と同じ情報がコマンドラインに表示されます。

Xdata_Add.lsp	拡張データを付加する
目的：	プリケーション名の登録と拡張データを付加します。
主要関数：	<entsel><car><entget><initget><getkword><cond><getreal><getint><getstring><list><append><entmod>
P1-257 で作成したアプリケーション名 [cost] に拡張データを付加します。	

```
(defun C:Xdata_Add( / apname old_ent kword data1 data2 add1 add2 adddata new_ent)
; 図面にはアプリケーション名として "cost" がすでに登録されています。
❶ (setq old_ent (car (entsel "¥n 拡張データを付加する図形を選択 : " ))) ; 図形を選択する
❷ (setq old_ent (entget old_ent)) ; 選択した図形の情報を得る
❸ (initget 0 "Real Integer String" ) ;initget 関数で、入力を [R] と [I] と [S] に制限する
❹ (setq kword (getkword "¥n 拡張データのタイプ [R= 実数 /I= 整数 /<S= 文字列 >]:" ))
❺ (cond ;3 つの条件によって、プログラムを分岐する
❻  ((= kword "Real" )                                   ;[R] が入力された時の処理
     (setq data1 (getreal "¥n 拡張データの値（実数）を入力 :" )  ; 実数の入力を促す
         code 1040)                                     ; 実数の拡張コードは 1040
     )
❼  ((= kword "Integer" )                                ;[I] が入力された時の処理
     (setq data1 (getint "¥n 拡張データの値（整数）を入力 :" )   ; 整数の入力を促す
         code 1070)                                     ; 整数の拡張コードは 1070
     )
❽  ((= kword "String" )                                 ;[S] が入力された時の処理
     (setq data1 (getstring "¥n 拡張データの値（文字列）を入力 :" ) ; 文字の入力を促す
         code 1000)                                     ; 文字の拡張コードは 1000
     )
  ) ;cond

❾ (setq xdata (list (list -3 (list "cost" (cons code data1))))) ; 追加する拡張データのリストを作成
❿ (setq new_ent (append old_ent xdata)) ; 元の図形に拡張データを追加する
⓫ (entmod new_ent) ; 図形情報を更新する（必須）
  (princ)
) ;end
```

Step1 ー①②

拡張データを付加する図形を選択し、図形情報を entget 関数で取得します。
（最後に append 関数でリストを追加し、entmod 関数で図形情報を更新するためです。）



Step2 ー③④⑤⑥⑦⑧

付加する拡張データのタイプ（実数、整数、文字）を指定します。
タイプによって対応する拡張コードを付加して、（拡張コード．値）のドット・ペアを作成します。

Step 3 ー⑨⑩⑪

作成した拡張データを元の図形に追加して、1 つのリストにします。
最後に図形情報を更新します。

番号	関数名	説　明
①	old_ent	選択した図形の図形名を取得して、変数 old_ent にセットします。
②	entget	entget 関数で選択した図形情報を取得します。
③	initget	initget 関数は入力をチェックする関数です。
④	getkword	getkword 関数の引数はプロンプトだけです。getkword は initget とペアで使います。initget 関数を使って指定してある値だけが getkword に入力できます。
⑤	cond	キーワードの文字によってプログラムを分岐します。
⑥	“Real”	kword が” R” であれば、data1 に実数の値が入り、拡張データのグループ コードとして code に <1040> をセットします。
⑦	“Integer”	kword が” I” であれば、data1 に整数の値が入り、拡張データのグループ コードとして code に <1070> をセットします。
⑧	“String”	kword が” S” であれば、data1 に文字列の値が入り、拡張データのグループ コードとして code に <1000> をセットします。
⑨	list -3	<-3> は拡張データの始まり を示すグループ コードです。
	list "cost"	アプリケーション名 [cost] に拡張データを加え、1 つのリストにします。
	cons	code と data1 でドット・ペアを作成し、リストに加えます。
⑩	append	複数のリストを 1 つのリストに結合する関数です。
⑪	entmod	図形情報を更新します。（必須です）

3 拡張データを取得する

entget 関数を使うことにより、図形の拡張データを取得できます。entget 関数は、図形の標準の定義データと、entget を呼び出すときにアプリケーションを指定した場合は Xdata を取得します。

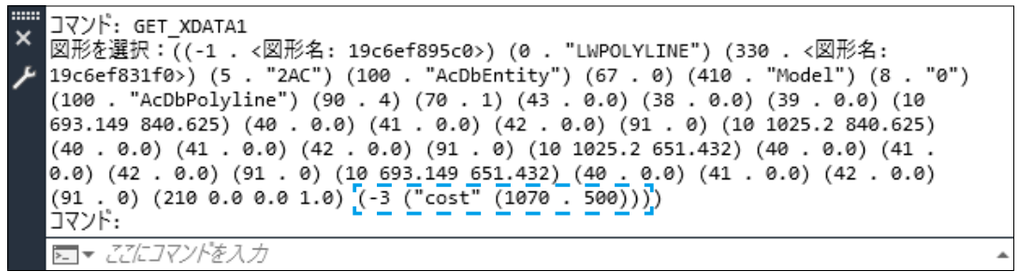
entget を使用して Xdata を取得する場合、拡張データの始まりはコード -3 で示されます。
コード -3 は、最初のグループ 1001 のリストの直前にあります。グループ 1001 には、取得した最初のアプリケーションの名前が含まれます。
もし、アプリケーション名が分からない場合は、ワイルドカード < * > が使えます。

```
(setq apname “*” )
(setq ent (car (entsel “¥n 拡張データを持つ図形を指示：” )))
(setq ldata (entget ent (list apname)))
(setq xdata (cadr (assoc -3 ldata)))
```

前ページで拡張データを付加した図形で確認してみます。

```
(defun C:get_xdata1( / en1 xdata)
; 図形を選択します
 (setq en1 (entsel "¥n 図形を選択："))
; 拡張データを含む全情報が取得できます
 (setq xdata (entget (car en1) '("*"))) ; ワイルドカード ("*") を使用
);end
```

コマンドラインに選択した図形の全データが表示されます。
最後の <-3> 以下が拡張データになります。（整数に <500> を入力した例）



get_xdata2.lsp	拡張データを取得（拡張データのみ取得）
目的：	拡張データの値だけを表示します。
主要関数：	<entsel><car><entget><asooc><cdr><if><progn><print>

```
(defun c:get_xdata2 ( / ent ent1 en en1 en2 xdata1 )
;P1-259 で付加した拡張データの情報を取得する（アプリケーション名を指定する場合）
❶ (setq ent (car (entsel "¥n 図形を選択： "))) ;< 図形名 : 7ef02c00> を取得する
❷ (if (/= ent nil) ;<nil> で無ければ、以下を実行する
    (progn
❸ (setq ent1 (entget ent '("cost"))) ; アプリケーション名 [cost] の図形を取得する
;ent1 の末尾の <(-3 ("cost" (1070 . 500)))> を assoc 関数で取り出す
❹ (setq en (cdr (assoc -3 ent1))) ;cdr 関数で 2 番目以降の情報を取得する (en)
;en の値は <(("cost" (1070 . 500)))>
❺ (if (/= en nil)                    ; <nil> で無ければ、以下を実行する
    (progn
❻ (setq en1 (car en)) ; 一番外側の () を取り、("cost" (1070 . 500)) を取得する
❼ (setq en2 (cdr en1)) ;"cost" 以降の ((1070 . 500)) を取得する
;en2 の値の <((1070 . 500))> から assoc 関数で (1070 . 500) を取り出す
❽ (setq xdata1 (cdr (assoc 1070 en2))) ;500 だけを取得する
;xdata1 の値は <500>
❾ (print xdata1) ;500 をコマンドラインに表示する
    );prog
  );if
 );prog
);if
);end
```

番号	関数名	説　明
①	entsel	拡張データが付加されている図形を選択します。
②	/= ent nil	選択されていれば、以下のプログラムを実行します。
③	entget	アプリケーション名が [cost] の情報を取得します。
④	assoc	アプリケーション名 (-3) 以下の情報を取得します。
⑤	/= en nil	情報が nil でなければ、以下のプログラムを実行します。
⑥	car en	(("cost" (1070 . 500)) の外側の () を car 関数で外します。
⑦	cdr en1	("cost" (1070 . 500)) の後半の (1070 . 500) を取得します。
⑧	(cdr (assoc 1070 en1))	ドット・ペア (1070 . 500) の後半の 500 を取得します。
⑨	print	拡張データ <500> をコマンドラインに表示します。

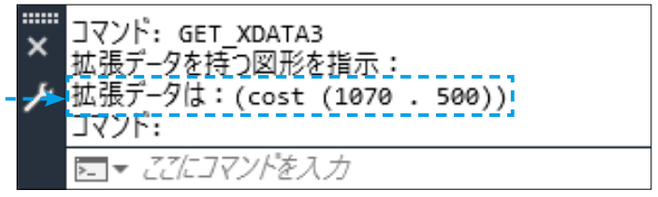
拡張データだけが、コマンドラインに表示されます。
500

get_xdata3.lsp	拡張データを取得（アプリケーション名と拡張データを取得）
目的：	アプリケーション名と拡張データの値を表示します。
主要関数：	<entsel><car><list><entget><asooc><cadr><if><prompt><progn><princ>

```
(defun c:get_xdata3( /  apname ent data1 xdata1)
   ;P1-259 で付加した拡張データの情報を取得する（アプリケーション名が判らない場合）
❶ (setq apname "*" ) ; ワイルド カードで検索するために変数にワイルド カード <*> をセット
❷ (setq ent (car (entsel "¥n 拡張データを持つ図形を指示： " ))) ; 図形名を取得する
❸ (setq data1 (entget ent (list apname))) ; 拡張データを持つドット・ペアのリストを取得する
;data1 の末尾の <(-3 ("cost" (1070 . 500)))> を assoc 関数で取り出す
❹ (setq xdata1 (cadr (assoc -3 data1))) ;(assoc -3 data1) の値は <(-3 ("cost" (1070 . 500)))>
;(assoc -3 data1) の値から <-3> 以降を cadr 関数で取り出す（この場合は <cdr> でも同じ）
;xdata1 の値は <("cost" (1070 . 500))>
❺ (if (= xdata1 nil) ; アプリケーション名の拡張データが <nil> かどうかをチェックする
❻ (prompt "¥n 拡張データがありません" )) ;<nil> であれば終了する
   (progn ;<nil> でなければ、以下を実行する
❼ (prompt  "¥n 拡張データは： " ) ; コマンド ラインに文字を表示する
    (princ xdata1)                ; コマンド ラインにアプリケーション名と拡張データを表示する
   ) ;progn
  ) ;if
) ;end
```

番号	関数名	説　明
①	"*"	* はワイルドカードです。アプリケーション名が分からない時に使用します。
②	entsel	拡張データが付加されている図形名を取得します。
③	list appname	アプリケーション名を示したグループ コードのリストを取得します。 拡張データはリストの末尾に表示されます。
④	xdata1	アプリケーション名の拡張データだけを取り出します。
⑤	if	アプリケーション名の拡張データがあるかどうかチェックします。
⑥	prompt	拡張データがなければ、そのまま終了します。
⑦	princ	アプリケーション名と拡張データをコマンドラインに表示します。

アプリケーション名と拡張データが、コマンド ラインに表示されます。
(cost (1070 . 500))



オブジェクトの操作

4 拡張データをフィルタする

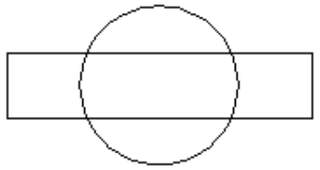
図面内にある拡張データ (XDATA) を持つブロック図形を次の条件で取得します。（P1-258）

①拡張データ (XDATA) を持つ全ての図形を取得する場合。

②拡張データ (XDATA) を持つブロック図形を取得する場合。

③アプリケーション名が <"light"> のブロック図形を取得する場合。

付加されている拡張データは、（< 電灯 > 文字列）、（<2012> 整数）、（<100.0> 実数）です。
またアプリケーション名は、<"light"> です。



①拡張データ (XDATA) を持つ全ての図形を取得する。

1.(setq ent (ssget "X" '((-3 ("*"))))) と入力します。（拡張データを持つ全図形を取得）
→ <Selection set: 19a>

2.(setq ent_len (sslength ent)) と入力します。（選択セットの数を取得）
→ 1

3.(setq ent1 (ssname ent 0)) と入力します。（選択セットの最初の図形を取得）
→ < 図形名 : 7ef03cb8>

4.(setq ent2 (entget ent1)) と入力します。（拡張データ付きの情報が表示されます。）
→ ((-1 . < 図形名 : 7ef03cb8>) (0 . "INSERT") (330 . < 図形名 : 7ef01cf8>) (5 . "2FF")
(100."AcDbEntity")(67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbBlockReference")(2 . "cost")
(10 1318.58 1420.44 0.0) (41 . 1.0) (42 . 1.0) (43 . 1.0) (50 .0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0)
(210 0.0 0.0 1.0) (-3 ("light" (1000 . " 電灯 ") (1070 . 2012) (1040 . 100.0))))

②拡張データ (XDATA) を持つブロック図形を取得する。

1.(setq ent (ssget "X" '((0 . "INSERT") (-3 ("*"))))) と入力します。（拡張データを持つブロックを取得）
→ <Selection set: 1ac>

2. ①の 2 から 4 を繰り返します。（同じデータが取得できます。）

③アプリケーション名が <"light"> のブロック図形を取得する。

1.(setq ent (ssget "X" '((0 . "INSERT") (-3 ("light"))))) と入力します。
（"light" の拡張データを持つブロックが取得できます。）
→ <Selection set: 1b0>

2. ①の 2 から 4 を繰り返します。（同じデータが取得できます。）

第 2 部　ダイアログ コントロール言語 (DCL)

第 1 章　ダイアログとタイルの構造

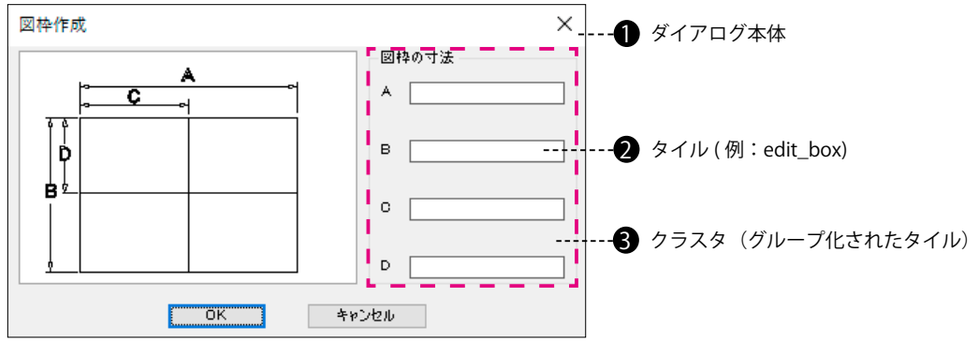
AutoCAD が提供するダイアログでは、ユーザーが入力する
テキスト ボックスやボタンを [タイル] と呼びます。

この章では、ダイアログ コントロール 言語 (DCL) の構造と
構成要素であるタイルについて解説します。

第 1 節　ダイアログボックスの構成

ユーザーはダイアログ ボックスを使用して、数値や文字の入力を行うことができます。
ダイアログ ボックスの外観は、ダイアログ コントロール言語 (DCL) ファイルで定義します。
そしてダイアログ ボックスの機能は、AutoLISP アプリケーションでコントロールします。

ダイアログ ボックスはダイアログ本体とタイル及びタイルをグループ化したクラスタで構成されています。

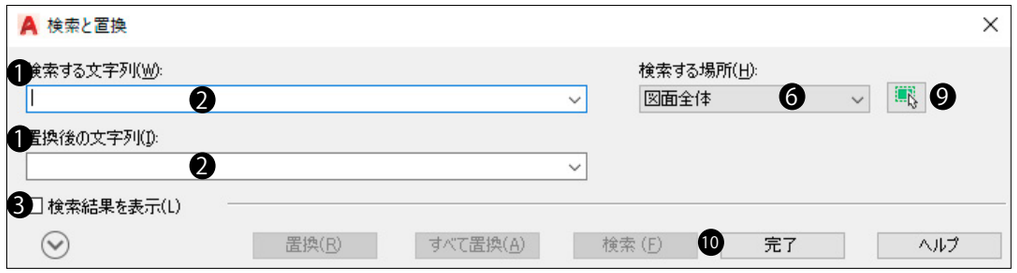


タイルには以下の種類があります。⑩の [Ok_Cancel] は他にもいくつかの組み合わせがあります。

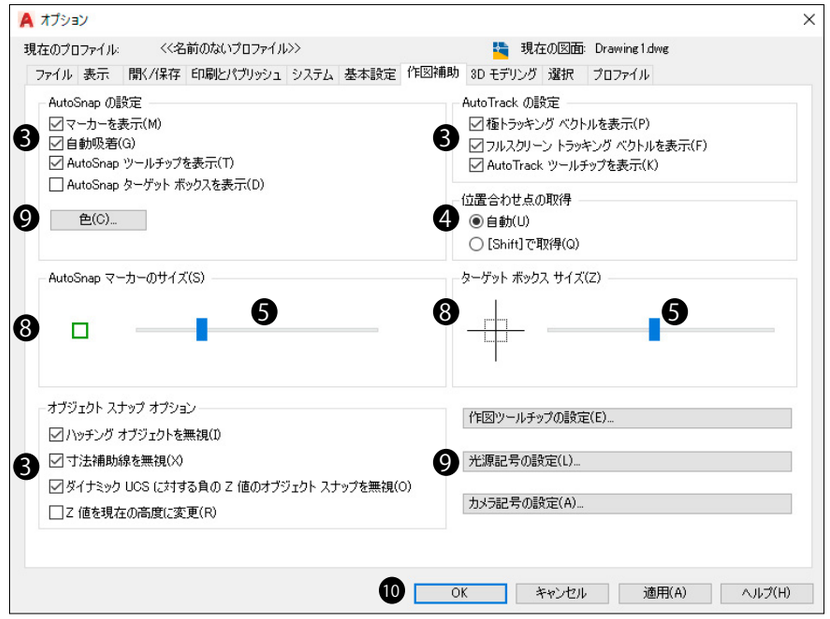
番号	タイル名	説　　明
①	text_tile	タイトルを作成したり、情報を提供するための文字列を表示します。
②	edit_box	1 行の文字を入力または編集するフィールド。
③	toggle_button	トグルは、真偽値（"0" か "1"）をコントロールします。
④	radio_button	ラジオ列またはラジオ行を構成する複数のボタンの 1 つ。
⑤	slider	スライダーのインジケータを左右にドラッグして値を取得します。
⑥	popup_list	テキスト又は矢印を選択すると、リストが開き選択肢が表示されます。
⑦	list_box	行単位で整列された文字列のリストです。複数の項目を選択できます。
⑧	image	ラベルではなく、グラフィックス イメージを表示する矩形です。
⑨	button	ダイアログ ボックスを閉じたり、別のダイアログを表示できます。
⑩	Ok_Cancel	このタイルは、base.dcl ファイルで定義されています。

1　AutoCAD が提供するダイアログ ボックス

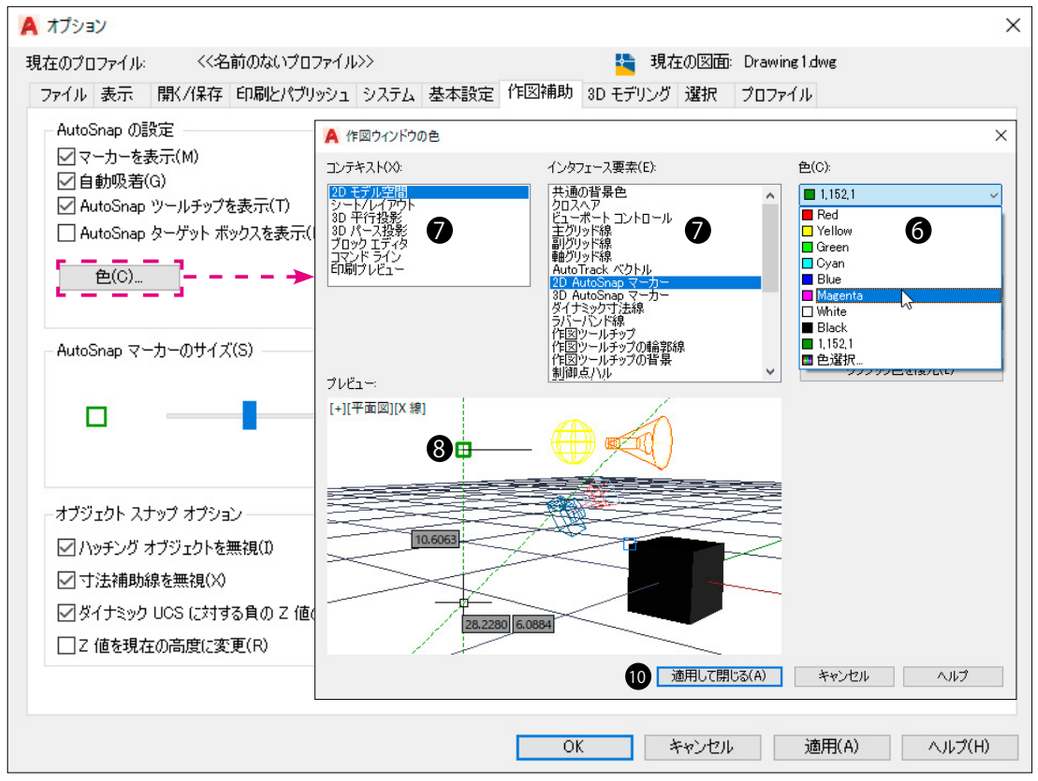
[編集] → [文字検索]



[ツール] → [オプション] → [作図補助] タブ

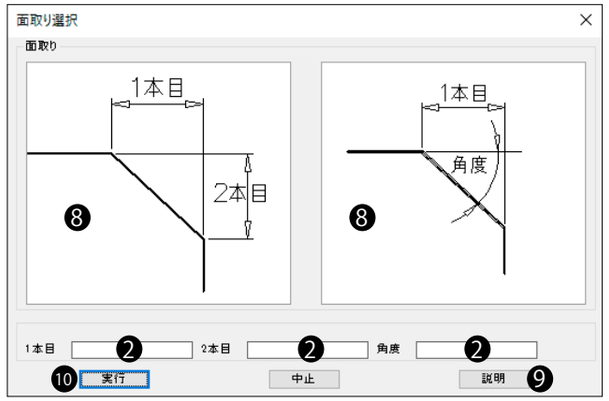


[ツール] → [オプション] → [作図補助] タブから 色(C)... ボタンを選択



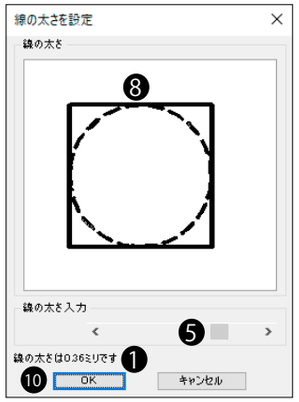
2 ユーザー作成のダイアログ ボックス

第 3 章 5 節



2 種類ある面取りのどちらかを選択して、面取りを行います。

第 3 章 6 節



線の太さの設定を画像で確認しながら行うことができます。

第 3 章 7 節

用紙の大きさと尺度を指定して、自動的に図枠を作図します。

第 3 章 10 節

図面内の図形に拡張データ (Xdata) を付加します。

第 4 章 4 節

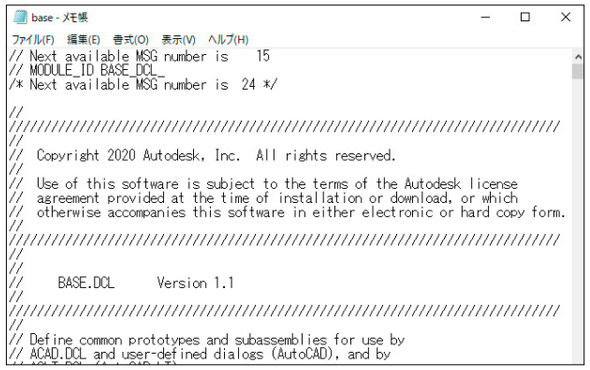
AutoCAD の [help] ボタンの位置に、ユーザーが作成した関数を割り当てています。
[help] ボタンは、このようにカスタマイズ可能です。

第 2 節 base.dcl ファイルと acad.dcl ファイル

AutoCAD には base.dcl ファイルと acad.dcl ファイルがあり、それぞれ AutoCAD のユーザーのサポートフォルダに格納されています。

① base.dcl ファイルには、基本的な定義済みのタイルとタイル タイプの DCL 定義が含まれています。このファイルには、よく使用するプロトタイプの定義も収められています。

base.dcl

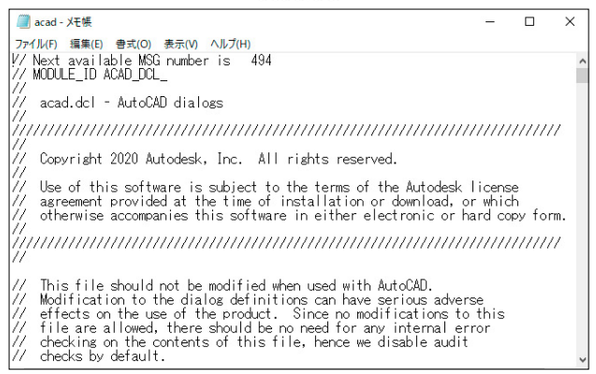


[ok_cancel_help] ボタンの記述

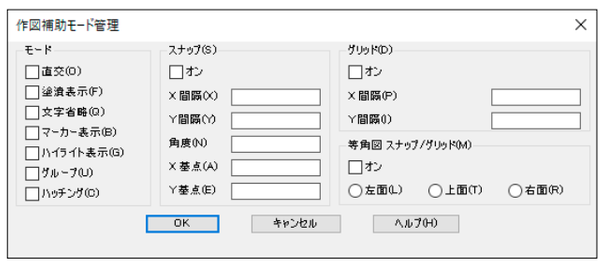
```
ok_cancel_help : column {
  : row {
    fixed_width = true;
    alignment = centered;
    ok_button;
    : spacer { width = 2; }
    cancel_button;
    : spacer { width = 2; }
    help_button;
  }}
```

② acad.dcl ファイルには、AutoCAD によって使用されるすべてのダイアログ ボックスの標準の定義が格納されています。

acad.dcl



[作図補助モード管理] のダイアログの記述



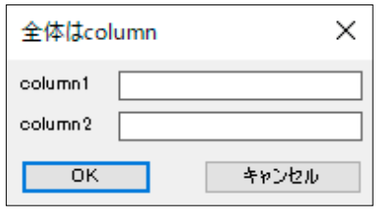

```
acad_snap : dialog {
    label = " 作図補助モード管理 ";
    : row {
      : column {
        : boxed_column {
          label = " モード ";
          : toggle {
            label = " 直交 (&O)";
            key = "ortho";
          }
          : toggle {
            label = " 塗潰表示 (&F)";
            key = "fill";
          }
          : toggle {
            label = " 文字省略 (&Q)";
            key = "qtext";
          }
          : toggle {
            label = " マーカー表示 (&B)";
            key = "blips";
          }
```

第 3 節　タイルのレイアウト

タイルは行または列にグループ化できます (総称してクラスタといいます)。
レイアウト上、クラスタは 1 つのタイルとして扱われます。行や列はボックスで囲むことができ、ラベルを付けることもできます (ボックスのないクラスタにラベルを付けることはできません)。

boxed_column	dialog
boxed_radio_column	radio_column
boxed_radio_row	radio_row
boxed_row	row
column	

① Column(<Column><Column>)

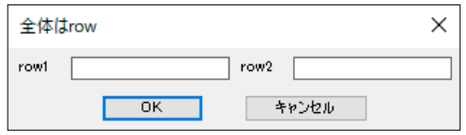


column
(edit)
(edit)

column2 :dialog

```
{
  label = " 全体は column";
  :column
  {
    :edit_box
    {
      label = "column1";
      key = "yoko_1";
      width = "";
      value = "";
    }
    :edit_box
    {
      label = "column2";
      key = "tate_1";
      width = "";
      value = "";
    }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

② Row(<Row><Row>)



row
(edit) (edit)

row2 :dialog

```
{
  label = " 全体は row";
  :row
  {
    :edit_box
    {
      label = "row1";
      key = "yoko_1";
      width = "";
      value = "";
    }
    :edit_box
    {
      label = "row2";
      key = "row2";
      width = "";
      value = "";
    }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

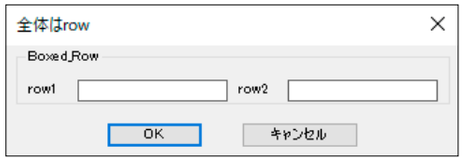

③ Column(<Column><Column>)

boxed_column
(edit)
(edit)

boxed_column2 :dialog

```
{
  label = " 全体は column";
  :boxed_column
  {
  label = "Boxed_Column";
    :edit_box
    {
      label = "column1";
      key = "yoko_1";
      width = "";
      value = "";
    }
    :edit_box
    {
      label = "column2";
      key = "tate_1";
      width = "";
      value = "";
    }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

④ Row(<Row><Row>)

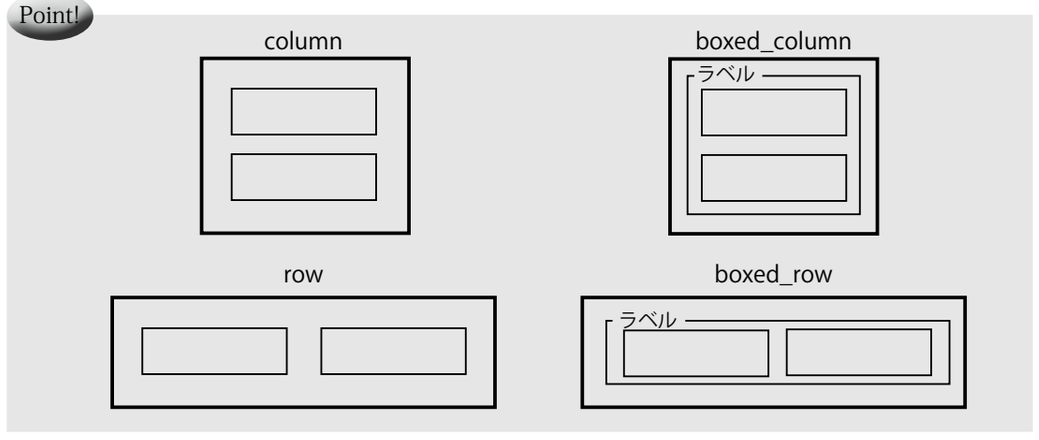


boxed_row
(edit) (edit)

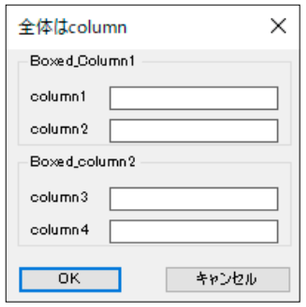
boxed_row2 :dialog

```
{
  label = " 全体は row";
  :boxed_row
  {
  label = "Boxed_Row";
    :edit_box
    {
      label = "row1";
      key = "yoko_1";
      width = "";
      value = "";
    }
    :edit_box
    {
      label = "row2";
      key = "tate_1";
      width = "";
      value = "";
    }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

Point!



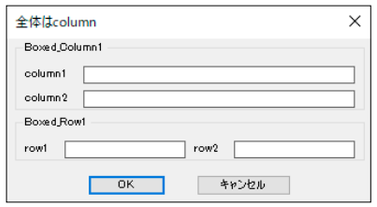
⑤ Column(<Column><Column>)




```
column
  boxed_column
    (edit)
    (edit)
  boxed_column
    (edit)
    (edit)
```

```
boxed_column4 :dialog
{
  label = " 全体は column";
  :column
  {
   :boxed_column
   {
     label = "Boxed_Column1";
     :edit_box
     {
       label = "column1";
       key = "yoko_1";
     }
     :edit_box
     {
       label = "column2";
       key = "tate_1";
     }
   }
   :boxed_column
   {
     label = "Boxed_column2";
     :edit_box
     {
       label = "column3";
       key = "yoko_2";
     }
     :edit_box
     {
       label = "column4";
       key = "tate_2";
     }
   }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

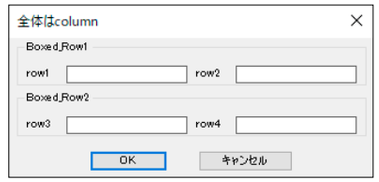
⑥ Column(<Column><Row>)




```
column
  boxed_column
    (edit)
    (edit)
  boxed_row
    (edit)  (edit)
```

```
boxed_column2_row2 :dialog
{
  label = " 全体は column";
  :column
  {
   :boxed_column
   {
     label = "Boxed_Column1";
     :edit_box
     {
       label = "column1";
       key = "yoko_1";
     }
     :edit_box
     {
       label = "column2";
       key = "tate_1";
     }
   }
   :boxed_row
   {
     label = "Boxed_Row1";
     :edit_box
     {
       label = "row1";
       key = "yoko_2";
     }
     :edit_box
     {
       label = "row2";
       key = "tate_2";
     }
   }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

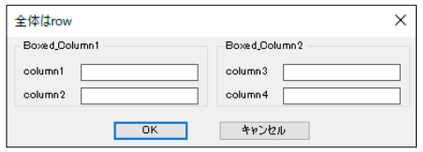

⑦ Column(<Row><Row>)




```
column
  boxed_row
    (edit)  (edit)
  boxed_row
    (edit)  (edit)
```

```
boxed_row4 :dialog
{
  label = " 全体は column";
  :column
  {
   :boxed_row
   {
     label = "Boxed_Row1";
     :edit_box
     {
       label = "row1";
       key = "yoko_1";
     }
     :edit_box
     {
       label = "row2";
       key = "tate_1";
     }
   }
   :boxed_row
   {
     label = "Boxed_Row2";
     :edit_box
     {
       label = "row3";
       key = "yoko_2";
     }
     :edit_box
     {
       label = "row4";
       key = "tate_2";
     }
   }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

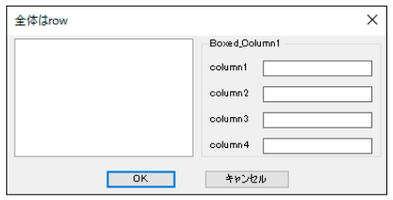
⑧ Row(<Column><Column>)




```
row
  boxed_column    boxed_column
    (edit)          (edit)
    (edit)          (edit)
```

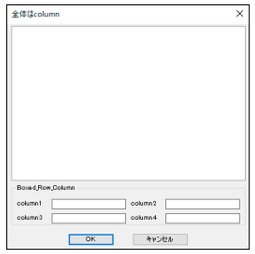
```
boxed_row2_column2 :dialog
{
  label = " 全体は row";
  :row
  {
   :boxed_column
   {
     label = "Boxed_Column1";
     :edit_box
     {
       label = "column1";
       key = "yoko_1";
     }
     :edit_box
     {
       label = "column2";
       key = "tate_1";
     }
   }
   :boxed_column
   {
     label = "Boxed_Column2";
     :edit_box
     {
       label = "column3";
       key = "yoko_2";
     }
     :edit_box
     {
       label = "column4";
       key = "tate_2";
     }
   }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

⑨ Row(<Row><Column>)




```
row
  (image)
  boxed_column
    (edit)
    (edit)
    (edit)
    (edit)
```

⑩ Column(<Column><Column><Row><Row>)




```
column
  (image)
  boxed_column
    row
      (edit)  (edit)
    row
      (edit)  (edit)
```

```
S1_Edit1 :dialog
{
  label = " 全体は row";
  :row
  {
    :icon_image
    {
      label = " パターン図 ";
      key = "image00";
    }
    :boxed_column
    {
      label = "Boxed_Column1";
      :edit_box
      {
        label = "column1";
        key = "yoko1";
      }
      :edit_box
      {
        label = "column2";
        key = "tate1";
      }
      :edit_box
      {
        label = "column3";
        key = "yoko2";
      }
      :edit_box
      {
        label = "column4";
        key = "tate2";
      }
    }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

```
S1_Edit2 :dialog
{
  label = " 全体は column";
  :column
  {
    :icon_image
    {
      label = " パターン図 ";
      key = "image00";
    }
    :boxed_column
    {
      label = "Boxed_Row_Column";
      :row
      {
        :edit_box
        {
          label = "column1";
          key = "yoko1";
        }
        :edit_box
        {
          label = "column2";
          key = "tate1";
        }
      }
      :row
      {
        :edit_box
        {
          label = "column3";
          key = "yoko2";
        }
        :edit_box
        {
          label = "column4";
          key = "tate2";
        }
      }
    }
  }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}
```

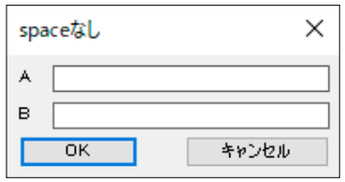

第 4 節 space と width

space や width 、height など、いくつかの属性はすべてのタイルに共通です。属性を割り当てないこともできます。多くの属性には、属性を割り当てないときに使用される既定値 (base.dcl で記述) があります。イメージの背景色など、特定の種類のタイルだけに使用される属性もあります。この属性を別の種類のタイルに割り当てようとした場合、エラーが表示されることがありますが、通常このような属性は無視されます。

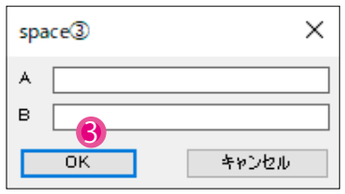
スペーサは、空白のタイルです。スペーサはレイアウトのためだけに使用され、隣接するタイルのサイズとレイアウトを変更します。PDB(プログラマブル ダイアログ ボックス) 機能は自動的に間隔を処理し、他のダイアログ ボックスと一貫性を保ちます。

1 縦方向の space

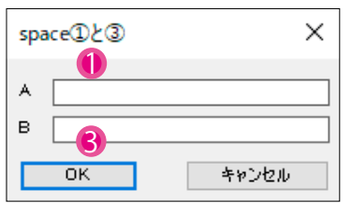
space 無し



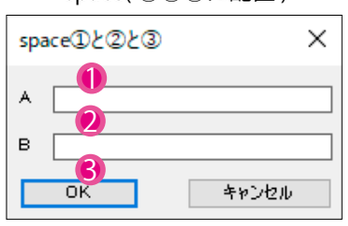
space(OK ボタンの上③に配置)



space(OK ボタンの上③、A の上①に配置)



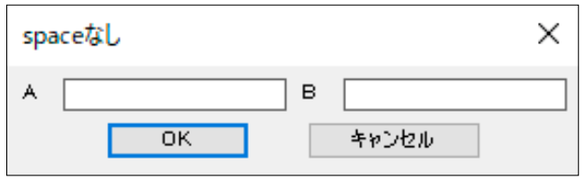
space(①②③に配置)



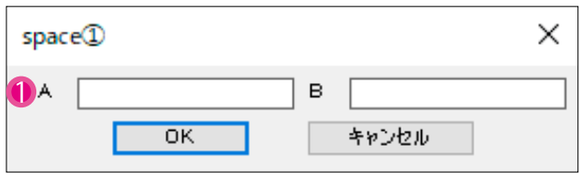

```
space4 :dialog
{
  label = "space ①と②と③ ";
  ❶ spacer;
  :column
  {
    :edit_box
    {
      label = "A";
      key = "yoko_1";
      width = "";
      value = "";
    }
  ❷ spacer;
    :edit_box
    {
      label = "B";
      key = "tate_1";
      width = "";
      value = "";
    }
  }
  /* OK or CANCEL */
  ❸ spacer;
  ok_cancel;
}
```

2 横方向の space

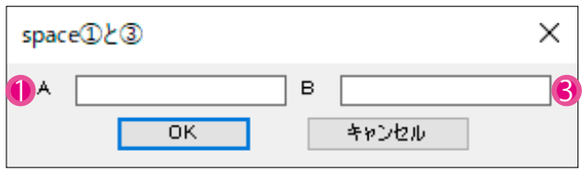
space 無し



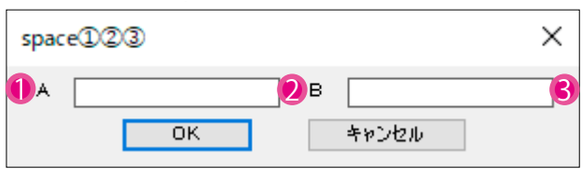
space(A の前に①を配置)



space(A の前に①、B の後に③を配置)



space(①②③に配置)




```
space8 :dialog
{
  label = "space ①②③ ";
  :row
  {
1 spacer;
    :edit_box
    {
     label = "A";
     key = "yoko_1";
     width = "";
     value = "";
    }
2 spacer;
    :edit_box
    {
     label = "B";
     key = "tate_1";
     width = "";
     value = "";
    }
3 spacer;
  }
  /* OK or CANCEL */
  ok_cancel;
}
```

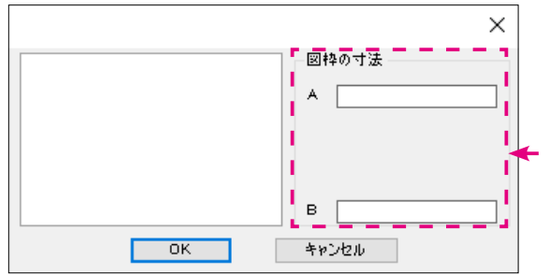
Point!

base.dcl で規定されている [ok_cancel] のスペースは以下の通りです。

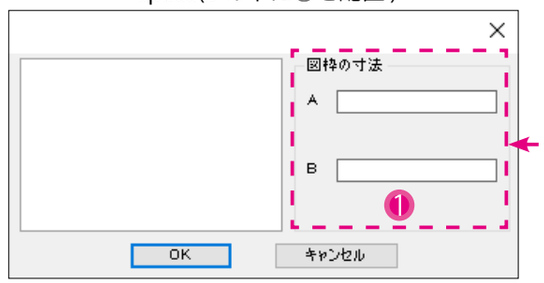
```
: row {
    fixed_width = true;          ←レイアウト時に幅が変化しません
    alignment = centered;        ←クラスタ内での水平または垂直位置が中央
    ok_button;                   ← OK ボタンの定義
    : spacer { width = 2; }      ← AutoCAD が割り当てる標準のスペース属性値
    cancel_button;               ← Cancel ボタンの定義
}
```


3 異なるタイルとの space

space 無し



space(B の下に①を配置)

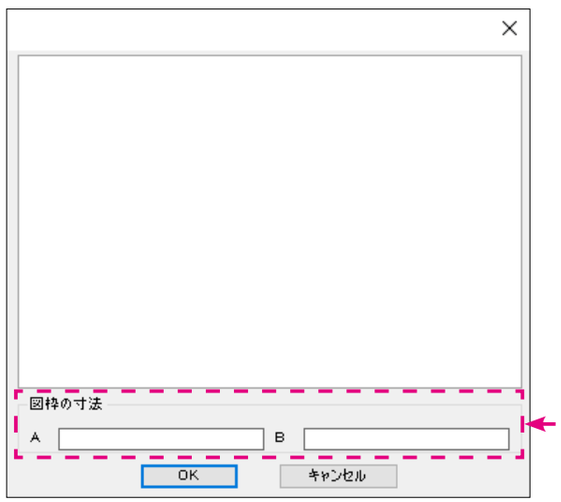



```
space22 :dialog
{
  label = "";
  :row
  {
    :icon_image
    {
      label = " パターン図 ";
      key = "image00";
    }
   :boxed_column
   {
    label = " 図枠の寸法 ";
    :edit_box
    {
      label = "A";
      key = "yoko1";
    }
    :edit_box
    {
      label = "B";
      key = "tate1";
    }
1   spacer;
   }
  }
  /* OK or CANCEL */
  ok_cancel;
}
```

Point!

スペースは PDB (プログラマブル ダイアログ ボックス) 機能で制約されていますから spacer を配置しても整列が不自然な場合は、下図のように row と column を入れ替えて レイアウトを変更した方が良い場合もあります。

:boxed_column を :boxed_row に変更する




```
   :boxed_row
   {
    label = " 図枠の寸法 ";
    :edit_box
    {
      label = "A";
      key = "yoko1";
    }
    :edit_box
    {
      label = "B";
      key = "tate1";
    }
   }
```

4 space_0 と space_1（base.dcl で定義済み）

spacer_0 は、通常は幅のないスペーサです。しかし、レイアウト時にタイル グループを引き伸す必要がある場合は、タイル グループのスペース挿入場所を示します。グループ内の spacer_0 タイルに正の幅を指定すると、それらすべてに等しい間隔が割り当てられます。

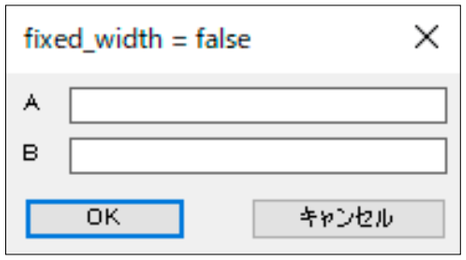
```
spacer_0 : spacer {
  height = 0;
  width = 0;
  horizontal_margin = none;
  vertical_margin = none;
}
```

spacer_1 タイルは、幅と高さが両方とも 1 のスペーサです。このタイルは、ユーザーに何とか見える程度の小さなスペーサに使用します。

```
spacer_1 : spacer {
  height = 1;
  width = 1;
  horizontal_margin = none;
  vertical_margin = none;
}
```

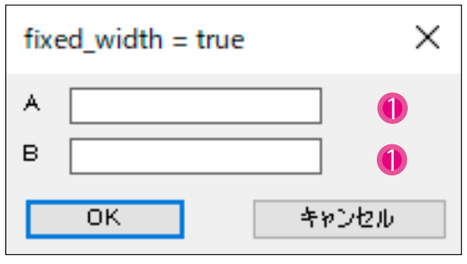
5 ixed_width = true,false

fixed_width = false



レイアウト時に幅が変化します。

fixed_width = true



レイアウト時に幅が変化しません。

```
width4 :dialog
{
  label = "fixed_width = true";
  :column
  {
    :edit_box
    {
      label = "A";
      key = "yoko_1";
①   fixed_width = true;
      value = "";
    }
    :edit_box
    {
      label = "B";
      key = "tate_1";
①   fixed_width = true;
      value = "";
    }
  }
   /* OK or CANCEL */
  spacer;
  ok_cancel;
}//end
```


第 5 節 タイルの属性

①属性のタイプ

属性の値は、次のいずれかのデータ タイプで指定します。

- 整数ータイルの幅や高さなど、長さを表す数値 (整数) で、文字幅または文字高さの単位で指定します。
- 実数ー同上ですが、1 未満の実数には小数点の前に 0 を付けます。たとえば、.1 ではなく 0.1 と指定します。

②クォーテーションで囲まれた文字列

文字列は、文字をクォーテーション マーク ("") で囲んで指定します。属性値では大文字と小文字が区別されます。たとえば、A1 は a1 と同じではありません。

文字列にクォーテーションを含めなければならない場合は、クォーテーションの前に円記号を付けます (¥")。

（例） "(set_tile ¥"text1¥" ¥" 面取りが選択されました。¥")"

DCL 文字列で使用できる制御文字	
制御文字	意 味
¥"	クォーテーション (文字列の内部)
¥¥	円記号
¥n	改行
¥t	水平タブ

③ DCL ファイルのコメント

DCL ファイルでは、2 つのスラッシュ (//) 以降はコメントとみなされ、// から行の終わりまでの文字は無視されます。

また、C 言語形式のコメントも認められており、その形式は /* コメント */ です。

左側の /* と右側の */ は同じ行になくてもかまいません。

（例）

```
:edit_box   /* 編集ボックス処理タイルスタート */
{
  label = "C";
  key = "yoko2";
  width = 6;
  value = "";
}       /* 編集ボックス処理タイル終了 */

/* OK or CANCEL */
spacer;
ok_cancel;
}//end
```

1 定義済み属性の概要

AutoCAD の PDB (プログラマブル ダイアログ ボックス) 機能には組み込みタイル (定義済みタイル) があり、それ単独で使用することも、より複雑なタイルの基礎として使用することもできます。これらの定義は、base.dcl 内でコメントで示されています。

ユーザーがアクティブなタイル (たとえばボタンなど) を選択すると、ダイアログ ボックスはそのダイアログ ボックスをコントロールしているアプリケーションに通知することによって応答します。定義済みのアクティブなタイルにはすべて、関連するアクションを指定できます。アクションの結果をユーザーに表示することも、内部だけで処理することもできます。

以下に、選択可能なアクティブなタイルを示します。

選択可能なアクティブタイル	
button	popup_list
edit_box	radio_button
image_button	slider
list_box	toggle

定義済み属性		
属性名	対 象	意味 (割り当てた場合、または true を指定した場合)
action	すべてのアクティブなタイル	AutoLISP のアクション式
alignment	すべてのタイル	クラスタ内での水平または垂直位置
allow_accept	edit_box、image_button、list_box	このタイルが選択されたときに is_default ボタンをアクティブにします
aspect_ratio	image、image_button	イメージの縦横比
big_increment	slider	スライダの移動量に対応する値の変化量
children_alignment	row, column、radio_row、radio_column、boxed_row、boxed_column、boxed_radio_row、boxed_radio_column	クラスタの子の位置合わせ
children_ixed_height	row, column、radio_row、radio_column、boxed_row、boxed_column、boxed_radio_row、boxed_radio_column	レイアウト時にクラスタの子の高さが変化しません
children_ixed_width	row, column、radio_row、radio_column、boxed_row、boxed_column、boxed_radio_row、boxed_radio_column	レイアウト時にクラスタの子の幅が変化しません
color	image、image_button	イメージの背景 (塗り潰し) 色
edit_limit	edit_box	ユーザが入力できる最大文字数
edit_width	edit_box、popup_list	タイルの編集 (入力) 部分の幅
fixed_height	すべてのタイル	レイアウト時に高さが変化しません
fixed_width	すべてのタイル	レイアウト時に幅が変化しません
fixed_width_font	list_box、popup_list	文字を固定幅フォントで表示します
height	すべてのタイル	タイルの高さ
initial_focus	Dialog	最初にフォーカスを設定するタイルのキー
is_bold	[文字]	太字で表示されます
is_cancel	Button	キャンセル キー (通常は [Esc]) を押すと、ボタンがアクティブになります。
is_default	Button	了解キー (通常は [Enter]) を押すと、ボタンがアクティブになります。
is_enabled	すべてのアクティブなタイル	初期状態でタイルが使用可能になります
is_tab_stop	すべてのアクティブなタイル	タイルがタブ停止位置になります
ワード	すべてのアクティブなタイル	アプリケーションが使用するタイル名
label	boxed_row、boxed_column、boxed_radio_row、boxed_radio_column、button、dialog、edit_box、list_box、popup_list、radio_button、text、toggle	タイルの表示ラベル
layout	slider	スライダが水平か垂直か
list	list_box、popup_list	リストに表示する初期値
max_value	slider	スライダの最大値
min_value	slider	スライダの最小値
mnemonic	すべてのアクティブなタイル	タイルのショートカット キー (ニモニック文字)
multiple_select	list_box	リスト ボックスで複数の項目を選択できるようにします
password_char	edit_box[編集ボックス]	edit_box に入力された文字を指定した文字でマスクします。
small_increment	slider	スライダの移動量に対応する値の変化量
tabs	list_box、popup_list	リスト表示のタブ停止位置
tab_truncate	list_box、popup_list	タブ停止位置を超える文字列を切り取ります。
value	text, アクティブなタイル (ボタンとイメージ ボタンを除く)	タイルの初期値
width	すべてのタイル	タイルの幅

第 6 節　個別タイルの構成

1 テキスト タイル (Text_Tile)

タイトルを作成したり、情報を提供するための文字列を表示します。

base.dcl で定義済みの既定値

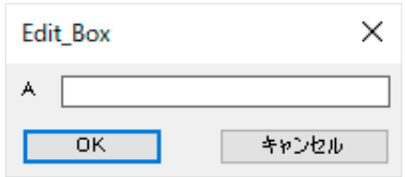
```
text : tile
{
  fixed_height    = true;
}
```

値 (Value) は、label と同様にテキストタイルに表示する文字列を指定

```
 : text
 {
   label = " 図形タイプ ";          ←タイル内に表示する文字を指定
   key = "text1";                   ←プログラムがこの text を参照するための名前
   width = 30;                      ←タイルの幅（文字列単位で指定）
 }
```

2 編集ボックス (Edit_Box)

1 行の文字を入力または編集するフィールド



base.dcl で定義済みの既定値

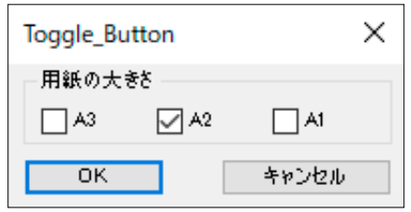
```
edit_box : tile
{
  fixed_height    = true;
  is_tab_stop     = true;
}
```

値 (Value) は、ASCII 文字列

```
 : edit_box
 {
   label = "A";                     ←タイル内の左側に表示する文字を指定
   key = "yoko_1";                  ←プログラムがこの edit_box を参照するための名前
   width = "";                      ←タイルの幅（文字列単位で指定）
   value = "";                      ←タイルの初期値（"" は空白の状態）
 }
```


3 トグル ボタン (Toggle)

トグルは、真偽値（"0" か "1"）をコントロールします。ユーザーがボタンを選択すると
チェックマークが表示または非表示されます。（1 つ 1 つは独立しています。）



base.dcl で定義済みの既定値

```
toggle : tile
{
  fixed_height    = true;
  is_tab_stop     = true;
}
```

値 (Value) は、"0" か "1" の文字列

```
 :toggle
 {
    label = "A3";                 ← toggle_button の右側に表示される文字
    key = "A3";                   ←プログラムがこの toggle_button を参照するための名前
 }
 :toggle
 {
    label = "A2";                 ← toggle_button の右側に表示される文字
    key = "A2";                   ←プログラムがこの toggle_button を参照するための名前
    value = "1";                  ← toggle の初期状態で、文字列が "1" の場合は選択されていることを表す
 }
 :toggle
 {
    label = "A1";                 ← toggle_button の右側に表示される文字
    key = "A1";                   ←プログラムがこの toggle_button を参照するための名前
 }
```

Point!

左のように記述した場合、内部データは右のように定義されます。

```
{                                    {
  label = "A3";          - - - - →     label = "A3";
  key = "A3";                          key = "A3";
}                                      fixed_height    = true;
                                       is_tab_stop     = true;
                                     }
```

4 ラジオ ボタン (Radio_Button)

クラスタ（グループ）の内、1 つのボタンしか選択できません。
（1 つを選択すると、他のボタンは自動的にオフ <"0"> になります。

base.dcl で定義済みの既定値

```
radio_button : tile
{
  fixed_height    = true;
  is_tab_stop     = true;
}
```

値 (Value) は、"0" か "1" の文字列

```
  :radio_row
  {
    label = " 用紙の大きさ ";
    key = "sel_yousi";
  {
    :radio_button
    {
      label = "A3";          ← radio_button の右側に表示される文字
      key = "A3";            ←プログラムがこの radio_button を参照するための名前
    }
  :radio_button
  {
    label = "A2";            ← radio_button の右側に表示される文字
    key = "A2";              ←プログラムがこの radio_button を参照するための名前
    value = "1";             ←この button の初期値は <On>
  }
 :radio_button
 {
   label = "A1";             ← radio_button の右側に表示される文字
   key = "A1";               ←プログラムがこの radio_button を参照するための名前
  }
```

Point!

左のように記述した場合、内部データは右のように定義されます。

```
{                          {
  label = "A3";    - - - - ->  label = "A3";
  key = "A3";                  key = "A3";
}                              fixed_height    = true;
                               is_tab_stop     = true;
                             }
```


5 ラジオ ボックス (Radio_Row & Column)

radio_row(column) には複数の radio_button が含まれますが、その内の 1 つしか選択できません。

値 (Value) は、現在選択されているラジオ ボタン（value が "1"）の key を表す文字列

```
  :radio_row
  {
    label = " 用紙の大きさ ";          ←ボックスに表示される文字
    key = "sel_yousi";                 ←プログラムがこのクラスタ（グループ）を参照するための名前
      {
        :radio_button
        {
          label = "A3";
          key = "A3";
        }
        :radio_button
        {
          label = "A2";
          key = "A2";
          value = "1";
        }
       :radio_button
       {
         label = "A1";
         key = "A1";
       }
```

base.dcl で定義済みの既定値

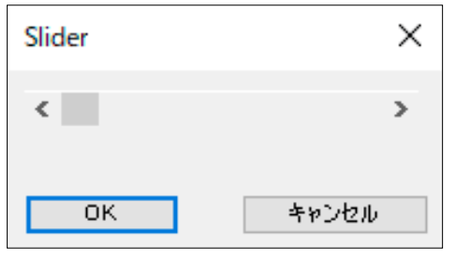
```
radio_row : radio_cluster
{
  horizontal_margin = none;
  vertical_margin = none;
  children_alignment = centered;
}
```

base.dcl で定義済みの既定値

```
boxed_radio_row : radio_cluster
{
  label = " ";
  boxed = true;
  children_alignment = centered;
}
```

6 スライダー (Slider)

スライダは、数値を獲得します。ユーザーはスライダーのインジケータを左右 (または上下) にドラッグすることによって、値を指定します。



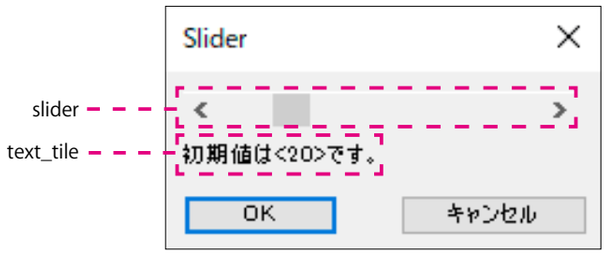
base.dcl で定義済みの既定値

```
slider : tile
{
  is_tab_stop    = true;
}
```

値 (Value) は、現在の（整数）値を表す文字列

```
  :slider
  {
    key = "slider01";              ←プログラムがこの slider を参照するための名前
    min_value = 0;                 ←スライダーが取得する値の最小値
    max_value = 100;               ←スライダーが取得する値の最大値
    small_increment = 10;          ←スライダーのインジケーターが動く最小単位
    big_increment = 20;            ←スライダーのインジケーターが動く最大単位
    width = 30;                    ←タイルの幅（文字列単位で指定）
    fixed_width = true;            ←タイルの幅を利用可能なスペースいっぱいに広げるかどうかを指定
    is_tab_stop = false;           ← [tab] を押してタイルを移動したときに、このタイルにフォーカスを設定
  }                                  するかどうかを指定（既定は true）
```

下の例では、"slider01" の値は、"20" です。

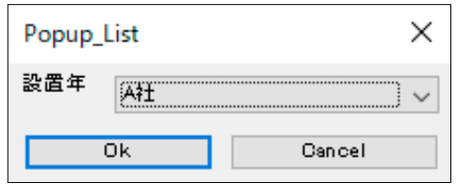


Point!

スライダーでは数値を表示するために
text_tile か edit_box を配置して
スライダーの数値を明示する必要があります。

7 ポップアップ リスト (Popup_List)

初めてダイアログ ボックスを表示したとき、ポップアップは閉じた状態で表示され、右側に下向きの矢印があることを除けばボタンと同じ形です。ユーザーがテキストまたは矢印を選択すると、リストが開いて選択肢が表示されます。



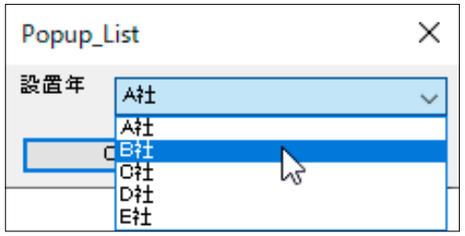
base.dcl で定義済みの既定値

```
popup_list : tile
{
  is_tab_stop    = true;
  fixed_height   = true;
}
```

値 (Value) は、"0" か "1" の文字列

```
  :popup_list
  {
    key = "list1";                 ←プログラムがこの list を参照するための名前
    label = "Popup_List";          ←タイル内に表示する文字を指定
    width = 15;                    ←タイルの幅（文字列単位で指定）
    value = "0";                   ←タイルの初期値（"0" はリストの第 1 番目を選択）
  }
```

下の例では、"list1" の値は、"1" です。



Point!

リストの 1 つを選択すると、それに対応する数値文字 ("0" or "1") が取得できます。

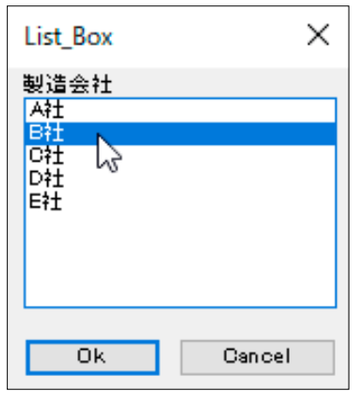
A 社を選択すると、<"0"> の値を取得します。
B 社を選択すると、<"1"> の値を取得します。
C 社を選択すると、<"2"> の値を取得します。
D 社を選択すると、<"3"> の値を取得します。
E 社を選択すると、 <"4"> の値を取得します。

8 リスト ボックス (List_Box)

行単位で整列された文字列のリストで、選択した行がハイライト表示されます。
popup_list と違って、複数の項目を選択することができます。

①リストから 1 つだけ選択する場合

Multiple_Select = FALSE;




```
base.dcl で定義済みの既定値
list_box : tile
{
  is_tab_stop    = true;
  height         = 10;
  width          = 10;
}
```

```
値 (Value) は、ゼロ個 ("")、または 1 つの整数を含む文字列
  : list_box
  {
   label ="List_Box";              ←タイル内に表示する文字を指定
   key = "list1";                  ←プログラムがこの list を参照するための名前
   height = 10;                    ←タイルの高さ（文字高さの単位で長さを表す整数または実数）
   width = 20;                     ←タイルの幅（文字列単位で指定）
   multiple_select = false;        ←複数の選択を許可しない
   value = "";                     ←タイルの初期値（"" は何も選択していない状態）
  }
```

上の例では、"list1" の値は、"1" です。

Point!

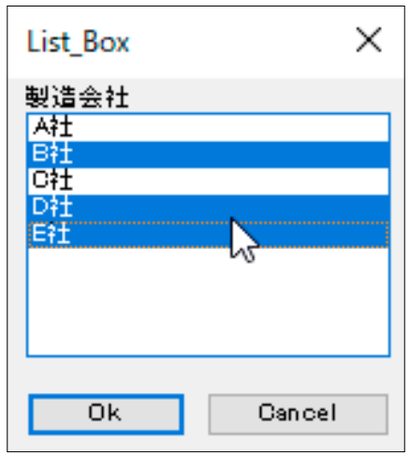
リストの 1 つを選択すると、それに対応する数値文字 "0" や "1" が取得できます。

A 社を選択すると、<"0"> の値を取得します。
B 社を選択すると、<"1"> の値を取得します。
C 社を選択すると、<"2"> の値を取得します。
D 社を選択すると、<"3"> の値を取得します。
E 社を選択すると、 <"4"> の値を取得します。

リストから、"0" や "1" に対応する文字 "A 社 "、"B 社 " を取得します。

②リストから複数選択する場合

Multiple_Select = TRUE;




```
base.dcl で定義済みの既定値
list_box : tile
{
  is_tab_stop    = true;
  height         = 10;
  width          = 10;
}
```

```
値 (Value) は、ゼロ個 ("")、またはそれ以上の整数を含む文字列
  : list_box
  {
   label ="List_Box";              ←タイル内に表示する文字を指定
   key = "list1";                  ←プログラムがこの list を参照するための名前
   height = 10;                    ←タイルの高さ（文字高さの単位で長さを表す整数または実数）
   width = 20;                     ←タイルの幅（文字列単位で指定）
   multiple_select = true;         ←複数の選択を許可する
   value = "";                     ←タイルの初期値（"" は何も選択していない状態）
  }
```

上の例では、"list1" の値は、"1 3 4" です。

Point!

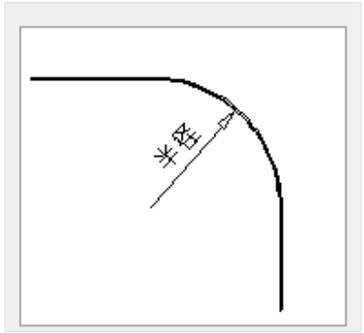
リストから複数を選択すると、全体を 1 つの数値文字 "0 1" が取得できます。

A 社だけを選択すると、<"0"> の値を取得します。
A 社、B 社、C 社を選択すると、<"0 1 2"> の値を取得します。
B 社、C 社、E 社を選択すると、<"1 2 4"> の値を取得します。

<"0 1 2"> のリストから read 関数で 1 つずつ取り出し、その数字に対応する文字をリストにしていきます。つまり、数値のリストから文字のリストに変換していきます。
結果は、("A 社 " "B 社 " "C 社 ") のリストになります。

9 イメージ (Image_Button)

イメージタイルは表示するだけでなく、key を割り当てて action 式を記述することができます。



base.dcl で定義済みの既定値

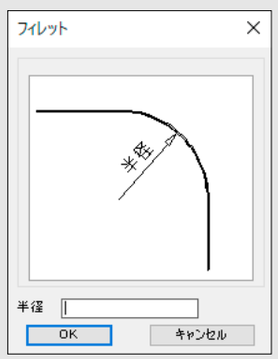
```
icon_image : image_button
{
  color          = 0;
  width          = 12;
  aspect_ratio   = 0.66;
  allow_accept   = true;
  fixed_height   = true;
  fixed_width    = true;
}
```

IMAGE

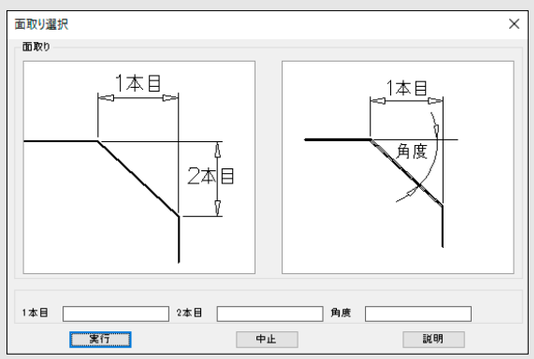
```
  :icon_image
  {
    key = "image01";          ←プログラムがこの image を参照するための名前
    width = 30;               ←タイルの幅（文字列単位で指定）
    height =15;               ←タイルの高さ（文字高さの単位で長さを表す整数または実数）
  }
```

Point!

左の図は、イメージを表示しているだけです。説明文よりイメージの方が
判りやすくなります。
右の図は、イメージを表示するだけでなく、どちらかのイメージを選択すれば
そのイメージに関連付けたプログラムを起動できます。



第 4 章 3 節

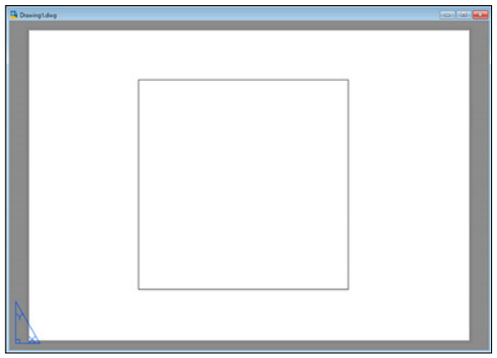


第 4 章 3 節

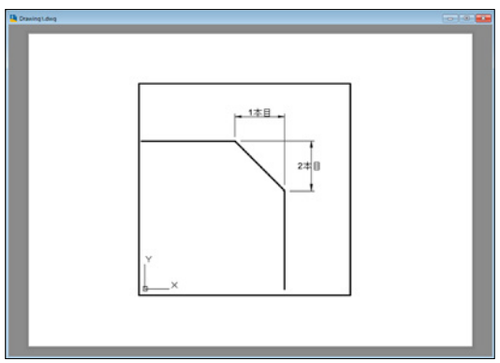
スライドファイルの作り方

Step1 ー AutoCAD のレイアウト空間（ペーパー空間）に移ります。
新規図面でも既存の図面でも構いません。

Step2 ー [レイアウト ビューポイント] パネル → [矩形] を選びます。（左図）
マウスで窓を 1 つ作成します。（右図）
縦横の比率は、ダイアログの縦横の比率と同じ程度にします。



Step3 ーレイアウトで、ペーパー空間からレイアウト ビューポート内のモデル空間に切り替えて、
作図します。（コマンド ラインから [mspace] と入力します。）

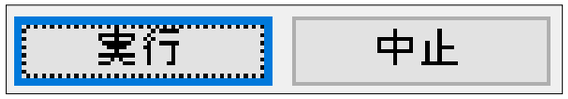


Step4 ー作図が完成すると、キーボードから <mslide> と入力します。
ファイル名を入力する時に、拡張子の <sld> は必要ありません。

10 ボタン (Button)

ボタンは、ダイアログ ボックスを終了したり、サブダイアログ ボックスに移るなど、ユーザーに直ちに表示されるアクションに適しています。

ダイアログ ボックスには [OK] ボタン (またはこれに相当するボタン) を配置し、ユーザーがボックスを使用したり見た後にそのボタンを押すようにします。多くのダイアログ ボックスには、ユーザーが変更しないでそのままダイアログ ボックスを終了できる、[キャンセル] ボタンもあります。



base.dcl で定義済みの既定値

```
button : tile
{
  fixed_height    = true;
  is_tab_stop     = true;
}
```

```
BUTTON
  : button
  {
    key = "button1;              ←プログラムがこの button を参照するための名前
    label = " 実行 ";            ←ボタンの上に表示する文字を指定
    is_tab_stop = true;          ← [tab] を押してタイルを移動したときに、このタイルにフォーカスを設定
    is_cancel = false;           ←キャンセルキー [Esc] を押した時に、このボタンは自動的に選択されない
    is_default = true;           ← [Enter] を押したときに、このタイルが自動的に選択されます
    width = 15;                  ←タイルの幅（文字列単位で指定）
    height = 10;                 ←タイルの高さ（文字高さの単位で長さを表す整数または実数）
  }
```

Point!

is_default = true;

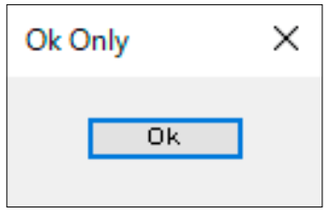
有効な値は、true または false です (既定は、false)。allow_accept 属性に true が指定されている edit_box、list_box、image_button の中では、ユーザーが了解キーを押したとき、または (リスト ボックスとイメージ ボタンの場合は) ダブルクリックしたときに、既定ボタンも選択されます。別のボタンにフォーカスが設定されている場合は、了解キーを押しても既定ボタンは選択されません。この場合、フォーカスが設定されているボタンが選択されます。

is_default 属性に true を指定できるのは、1 つのダイアログ ボックスについて 1 つのボタンだけです。

第 7 節 AutoCAD の Ok_Cancel ボタン

1 AutoCAD が提供する Ok_only ボタン

ok_only;



ユーザーが記述する場合
ok_only; のみ
ok_button key = "accept"

[ok_only;] は、base.dcl では以下のように記述されています。

```
  ok_only : column
  {
    fixed_width = true;          ←レイアウト時に幅が変化しません
    alignment = centered;        ←クラスタ内での水平または垂直位置が中央
    : ok_button                  ← OK ボタンの定義
    {
    is_cancel = true;            ←キャンセル キー ( 通常は [Esc]) を押すと、ボタンがアクティブ
    }
  }//end
```

2 AutoCAD が提供する Ok_cancel ボタン

ok_cancel;

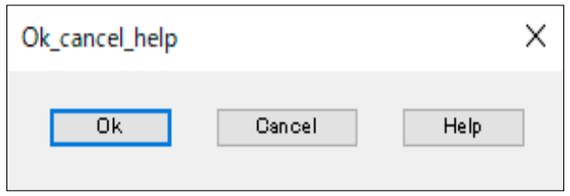
ユーザーが記述する場合
ok_cancel; のみ
ok_button key = "accept"
cancel_button key = "Cancel"

[ok_cancel;] は、base.dcl では以下のように記述されています。

```
  ok_cancel : column
  {
    : row
    {
      fixed_width = true;        ←レイアウト時に幅が変化しません
      alignment = centered;      ←クラスタ内での水平または垂直位置が中央
      ok_button;                 ← OK ボタンの定義
      : spacer { width = 2; }    ← AutoCAD が割り当てる標準のスペース属性値
      cancel_button;             ← Cancel ボタンの定義
    }
  }//end
```

3 AutoCAD が提供する Ok_cancel_help ボタン

ok_cancel_help;



ユーザーが記述する場合
ok_cancel_help; のみ
ok_button key = "accept"
cancel_button key = "Cancel"
help_button key = "help"

[ok_cancel_help;] は、base.dcl では以下のように記述されています。

```
    : row {
        fixed_width = true;              ←レイアウト時に幅が変化しません
        alignment = centered;            ←クラスタ内での水平または垂直位置が中央
        ok_button;                       ← OK ボタンの定義
        : spacer { width = 2; }          ← AutoCAD が割り当てる標準のスペース属性値
        cancel_button;                   ← Cancel ボタンの定義
        : spacer { width = 2; }          ← AutoCAD が割り当てる標準のスペース属性値
        help_button;
    }
  }//end
```

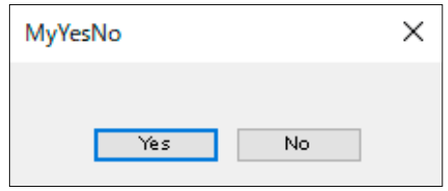
Point!

[OK] ボタン、[Cancel] ボタン、[Help] ボタンは AutoLISP では
一例として以下のように記述します。

① (action_tile "accept" "(done_dialog 1)")
　→ done_dialog の status（フラッグ）に <1> をセットして終了する
② (action_tile "cancel" "(done_dialog 0)")
　→ done_dialog の status（フラッグ）に <0> をセットして終了する
③ (action_tile "help" "(コールバック関数など)")
　→ダイアログを終了せずに、コールバック関数（サブ関数）を起動する

第 8 節 ユーザー定義の Ok_Cancel ボタン

1 ユーザーが作成する Ok_cancel ボタン



ユーザーが [Ok]、[Cancel] ボタンを配置するには、button タイルに記述します。

```
 MyYesNo : dialog
 {
  key = "Title";                    ←プログラムがこのダイアログのタイトルを参照するための名前
  label = "MyYesNo";                ←ボタンの上に表示する文字を指定
  spacer;                           ←スペースを配置
  : row {
   fixed_width = true;              ←レイアウト時に幅が変化しません
   alignment = centered;            ←クラスタ内での水平または垂直位置が中央
   : button {
    key = "accept";                 ←プログラムがこの button を参照するための名前
    label = "&Yes";                 ←タイルのショートカット キー ( ニモニック文字 )
    is_default = true;              ←了解キー ( 通常は [Enter]) を押すと、ボタンがアクティブ
    width = 8;                      ←タイルの幅（文字列単位で指定）
    fixed_width = true;             ←レイアウト時に幅が変化しません
   }
   : button {
    key = "Cancel";                 ←プログラムがこの button を参照するための名前
    label = "&No";                  ←タイルのショートカット キー ( ニモニック文字 )
    is_cancel = true;               ←キャンセル キー ( 通常は [Esc]) を押すと、ボタンがアクティブ
    width = 8;                      ←タイルの幅（文字列単位で指定）
    fixed_width = true;             ←レイアウト時に幅が変化しません
   }
  }
 }//end
```

Point!

[Ok]、[Cancel] ボタンのラベルを変更するためには、AutoCAD が提供する
<Ok_cancel;> ではなく、[button] を配置して、そのラベルに記述します。
そのボタンを <Ok> ボタンにするには、<key = "accept";>、<Cancel> ボタン
にするには <key = "Cancel";> と記述します。

2 ユーザーが作成する Ok_cancel_help ボタン

ユーザーが [Ok]、[Cancel] ボタンを配置するには、button タイルに記述します。

```
MyOkCancelHelp : dialog
{
  key = "Title";                    ←プログラムがこのダイアログのタイトルを参照するための名前
  label = "MyOkCancelHelp";         ←ボタンの上に表示する文字を指定
  spacer;                           ←スペースを配置
  : row
  {
    :spacer { width = 1; }          ←スペースを配置
    : button
    {
      label = " 処理実行 ";         ←ボタンの上に表示する文字を指定
      is_default = true;            ←了解キー ( 通常は [Enter]) を押すと、ボタンがアクティブ
      key = "accept";               ←プログラムがこの button を参照するための名前
      width = 8;                    ←タイルの幅（文字列単位で指定）
      fixed_width = true;           ←レイアウト時に幅が変化しません
    }
    :spacer { width = 1; }          ←スペースを配置
    : button
    {
      label = " 処理中止 ";         ←ボタンの上に表示する文字を指定
      is_cancel = true;             ←キャンセル キー ( 通常は [Esc]) を押すと、ボタンがアクティブ
      key = "Cancel";               ←プログラムがこの button を参照するための名前
      width = 4;                    ←タイルの幅（文字列単位で指定）
      fixed_width = true;           ←レイアウト時に幅が変化しません
    }
    :spacer { width = 1; }          ←スペースを配置
    : button
    {
      label = " 説明 ";             ←ボタンの上に表示する文字を指定
      key = "help";                 ←プログラムがこの button を参照するための名前
      width = 4;                    ←タイルの幅（文字列単位で指定）
      fixed_width = true;           ←レイアウト時に幅が変化しません
    }
    :spacer { width = 1; }          ←スペースを配置
  }
}//end
```


第 2 章 タイルの機能とコールバック関数

各タイルにはそれぞれの役割があり、機能も異なります。

この章では、ユーザーの目的に応じたタイルの使い分けと
そのコントロールの仕方について解説します。

第 1 節　ダイアログ ボックスの開始関数と終了関数

ダイアログ ボックスの 開始関数と終了関数の概要を、次の表に示します。

ダイアログ ボックス開始関数と終了関数		
	関数名	説 明
①	(load_dialog　dcl ファイル名)	DCL ファイルをロードします。
②	(new_dialog　ダイアログ名　dcl_id)	新しいダイアログ ボックスを開始し、初期化します。 既定のアクションを指定することもできます。
③	(start_dialog)	ダイアログ ボックスを表示し、ユーザー入力の受け入れを開始します。
④	(done_dialog [フラッグ <status>)	ダイアログ ボックスを終了します。
⑤	(unload_dialog　dcl_id)	DCL ファイルをロード解除します。

AutoLISP 関数でダイアログを宣言します。

(defun c:S_Edit1 (/ dialog-box dcl_id)

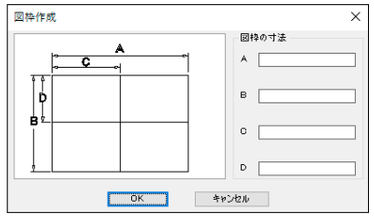
① load_dialog を呼び出し、DCL ファイルをロードします。

→ (setq dcl_id (load_dialog "S_Edit1.dcl"))

② new_dialog を呼び出し、特定のダイアログ ボックスを表示します。

→ (new_dialog "S_Edit1" dcl_id)

S_Edit1.dcl



- set_tile 関数
- action_tile 関数
- get_tile 関数
- mode_tile 関数

- リストの配置 (start_list、add_list、end_list)
- image の配置 (start_image、fill_image、slide_image、end_image)
- AutoLISP による計算処理
- サブ関数への分岐処理

③ start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。

→ (setq dialog-box (start_dialog))

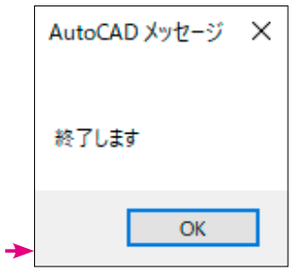
④ unload_dialog を呼び出して DCL ファイルをロード解除します。

→ (unload_dialog dcl_id)

[Ok] ボタンが押された時は、システム変数 [DIASTAT] に <1> がセットされます。
[Cancel] ボタンが押された時は、<0> がセットされます。

⑤もし [Ok] ボタンが押された時は、以下のプログラムを実行します。

→ (if(= (getvar "DIASTAT") 1) (alert "¥n 終了します。"))



③以降は次の方法もあります。
[Ok] ボタンで終了したときは、"done_dialog" のフラッグ (status) に <1> をセットします。

→ (action_tile "accept" "(done_dialog 1)")

[cancel] ボタンで終了したときは、"done_dialog" のフラッグ (status) に <0> をセットします。

→ (action_tile "cancel" "(done_dialog 0)")

③ start_dialog を呼び出して、コントロールをダイアログ ボックスに切り替えます。

→ (setq chkdia (start_dialog))

④ unload_dialog を呼び出して DCL ファイルをロード解除します。

→ (unload_dialog dcl_id)

⑤もし、(done_dialog) のフラッグ (status) が <1> であれば、以下のプログラムを実行します。

→ (if(= chkdia 1)(alert "¥n 終了します。"))

第 2 節　タイル処理関数と属性処理関数

DCL タイル処理関数と属性処理関数の概要を、次の表に示します。

タイル処理関数と属性処理関数		
	関数名	説 明
①	(set_tile　"key"　" 値 <value>")	ダイアログ ボックスのタイルの値を設定します。
②	(get_tile　"key")	ダイアログ ボックスのタイルの現在の値を取得します。
③	(mode_tile　"key"　値 <value>)	ダイアログ ボックスのタイルのモードを設定します。
④	(action_tile　"key"　" 処理内容 ")	ユーザーがダイアログ ボックスで特定のタイルを選択したときのアクションを割り当てます。

上記①から④は (new_dialog) と (start_dialog) の間に記述します。

```
(defun c:S_Edit1 ( / dialog-box dcl_id)
;<< ダイアログを読み込み、表示する。>>
  (setq dcl_id (load_dialog "S_Edit1"))
  (new_dialog "S_Edit1" dcl_id)
```

Point!

```
;<< スライドを読み込み、セットする >>・image の配置 (start_image、fill_image、slide_image、end_image)
  (setq x1 0 y1 0)                          ; スライドを表示する左上の原点を指定する
  (setq x2 (dimx_tile "image00"))           ; スライドを表示する X の幅を指定する
  (setq y2 (dimy_tile "image00"))           ; スライドを表示する Y の幅を指定する
  (start_image "image00")                   ;"image00" にスライドを読み込む
  (fill_image x1 y1 x2 y2 -2)               ; 背景色を現在の AutoCAD と同色にする
  (slide_image x1 y1 x2 y2 "S_Edit1")       ; スライド (S_Edit1) を表示する
  (end_image)                               ; スライドの読み込みを終了
```

Point!

```
;<< 各タイル の初期設定 >>・set_tile 関数
  (set_tile "yoko1" "")                     ;"yoko1" の初期値を空白にセットする
  (set_tile "yoko2" "")                     ;"yoko2" の初期値を空白にセットする
  (set_tile "tate1" "")                     ;"tate1" の初期値を空白にセットする
  (set_tile "tate2" "")                     ;"tate2" の初期値を空白にセットする
```

Point!

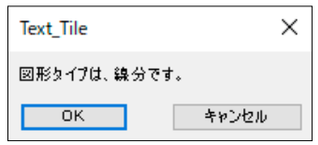
```
;<< edit_box 処理 yoko1 に値を受け取る >>・action_tile 関数・get_tile 関数・mode_tile 関数
  (action_tile "yoko1"                      ;"yoko1" の action を起こしたとき
    (strcat
      "(setq yoko1 (get_tile ¥"yoko1¥"))"   ;"yoko1" の値を変数 [yoko1] にセットする
      "(mode_tile¥"tate1¥" 2)"              ;"tate1" にカーソルのフォーカスを移す
    )
  )
        ・
      (中略)
        ・
;<<start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替える。>>
  (setq dialog-box (start_dialog))
;<<unload_dialog を呼び出して DCL ファイルをロード解除する。>>
  (unload_dialog dcl_id)
  (if(= (getvar "DIASTAT") 1) (alert "¥n 終了します。"))
)
```

❶ Set_Tile

目 的	ダイアログ ボックスの値を設定します。
書 式	(set_tile "key" " 値 <value>")

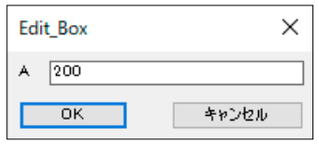
Text	key = "text1"　　セットする値 (value) = " 図形タイプは、線分です。"

(set_tile "text1" " 図形タイプは、線分です。") - - - →



Edit_Box	key = "yoko_1"　　セットする値 (value) = "200"

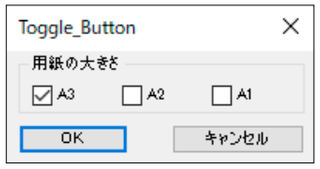
(set_tile "yoko_1" "200") - - - →



整数や実数、角度をセットする場合は、文字列に変換してセットします。

Toggle_Button	key = "A3"　　セットする値 (value) = "1"
	key = "A2"　　セットする値 (value) = "0"
	key = "A1"　　セットする値 (value) = "0"

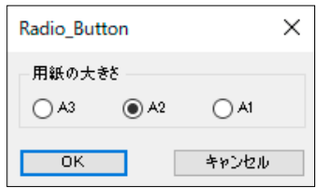
(set_tile "A3" "1")
(set_tile "A2" "0")
(set_tile "A1" "0") - - - →



それぞれのチェック ボタンは独立しています。
"A3" をチェックしたとき、他の 2 つは自動的にオフ <"0"> になりません。

Radio_Button	key = "A3"　　セットする値 (value) = "0"
	key = "A2"　　セットする値 (value) = "1"
	key = "A1"　　セットする値 (value) = "0"

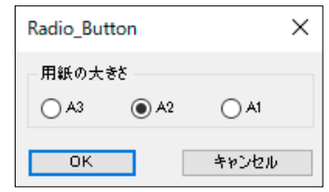
(set_tile "A2" "1") - - - →



"A2" を "1" にセットすると、他の 2 つ（"A3" と "A1"）は自動的にオフ <"0"> にセットされます。

Radio_Row	key = "sel_yousi"　セットする値 (value) = "A2"

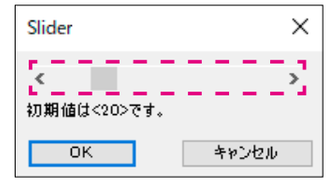
(set_tile "sel_yousi" "A2") - - - →



Radio_Row（または、Radio_Column）は、そのクラスタ（グループ）の Radio_Button の key 名を指定します。

Slider	key = "slide01"　　セットする値 (value) = "20"

(set_tile "slide01" "20") - - - →

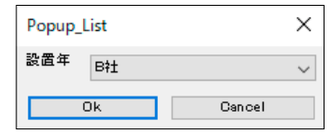


スライダーの値は、text_tile や edit_box に表示させます。

Popup_List	key = "list1"　　セットする値 (value) = "1"

list1 のデータは（"A 社 " "B 社 " "C 社 " "D 社 " "E 社 "）

(set_tile "list1" "1") - - - →

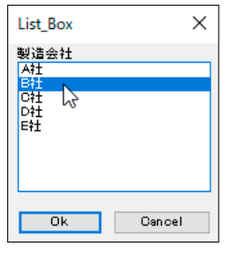


リストの値は、リストの具体的な値（名前）でなく、"0" から始まるリスト番号（文字）で指定します。

List_Box（1 つだけ）	key = "list1"　　セットする値 (value) = "1"

list1 のデータは（"A 社 " "B 社 " "C 社 " "D 社 " "E 社 "）

(set_tile "list1" "1") - - - →

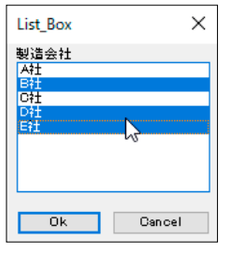


リストの値は、リストの具体的な値（名前）でなく、"0" から始まるリスト番号（文字）で指定します。

List_Box（複数）	key = "list1"　　セットする値 (value) = "1 3 4"

list1 のデータは（"A 社 " "B 社 " "C 社 " "D 社 " "E 社 "）

(set_tile "list1" "1 3 4") - - - →

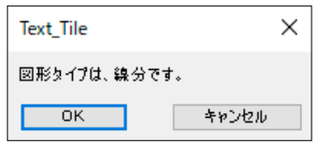


リストの値は、リストの具体的な値（名前）でなく、"0" から始まるリスト番号（文字）で指定します。

❷ Get_Tile

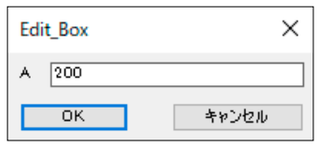
目 的	ダイアログ ボックスのタイルの現在の値を取得します。
書 式	(get_tile "key")

Text	key = "text1"　　取得する値 (value) = " 図形タイプは、線分です。"



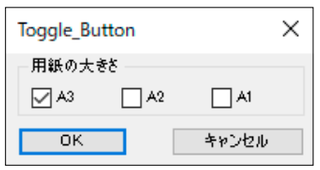
(setq a (get_tile "text1"))
→ a = " 図形タイプは、線分です。"

Edit_Box	key = "yoko_1"　　取得する値 (value) = "200"



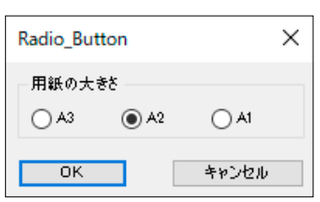
(setq a (get_tile "yoko_1")
→ a = "200"

Toggle_Button	key = "A3"　　取得する値 (value) = "1"
	key = "A2"　　取得する値 (value) = "0"
	key = "A1"　　取得する値 (value) = "0"



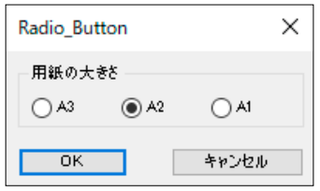
(setq a (get_tile "A3") → a = "1"
(setq b (get_tile "A2") → b = "0"
(setq c (get_tile "A1") → c = "0"

Radio_Button	key = "A3"　　取得する値 (value) = "0"
	key = "A2"　　取得する値 (value) = "1"
	key = "A1"　　取得する値 (value) = "0"



(setq a (get_tile "A3") → a = "0"
(setq b (get_tile "A2") → b = "1"
(setq c (get_tile "A1") → c = "0"

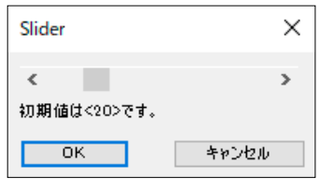
Radio_Row	key = "sel_yousi"　取得する値 (value) = "A2"



(setq a (get_tile "sel_yousi")
→ a = "A2"

Radio_Row（または、Radio_Column）は、そのクラスタ（グループ）の Radio_Button の key 名を取得します。

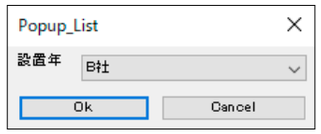
Slider	key = "slider01"　取得する値 (value) = "20"



(setq a (get_tile "slider01")
→ a = "20"

Popup_List	key = "list1"　取得する値 (value) = "1"

list1 のデータは（"A 社 " "B 社 " "C 社 " "D 社 " "E 社 "）

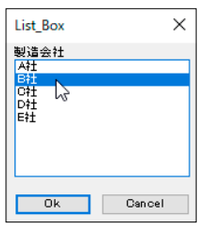


(setq a (get_tile "list1")
→ a = "1"

☞"1" から "B 社 " を取得する方法は、P2-132 の⑥を参考

List_Box（1 つ）	key = "list1"　取得する値 (value) = "1"

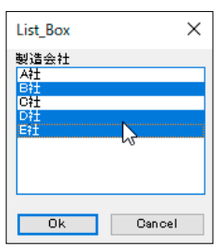
list1 のデータは（"A 社 " "B 社 " "C 社 " "D 社 " "E 社 "）



(setq a (get_tile "list1")
→ a = "1"

☞"1" から "B 社 " を取得する方法は、P2-141 の③を参考

List_Box（複数）	key = "list1"　取得する値 (value) = "1 3 4"



(setq a (get_tile "list1")
→ a = "1 3 4"

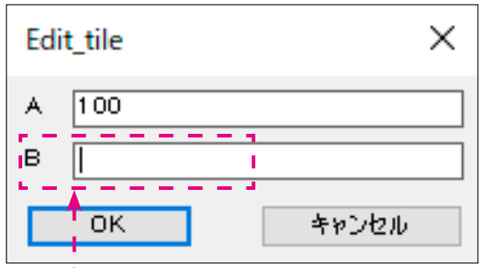
☞"1 3 4" から "B 社 " "D 社 " "E 社 " を取得する方法は、P2-147 の⑧を参考

❸ Mode_Tile

目 的	ダイアログ ボックスのタイルのモードを設定します。
書 式	(mode_tile "key" 値 <value>)

mode_tile の mode のコード	
値 <value>	説　明
0	タイルを使用可能にします。
1	タイルを使用禁止にします。
2	タイルにフォーカスを設定します。
3	編集ボックスの内容を選択します。
4	イメージのハイライト表示のオンとオフを切り替えます。

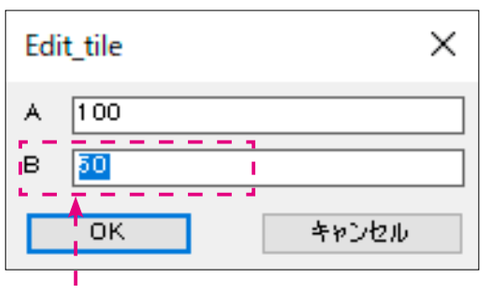
Edit_Box	mode の値 <value> が <2> のケース



A に入力 <100> して、エンターキー（改行）を押すと、カーソルが次の B のボックスに移ります。

```
;<< edit_box 処理 tate<A> に値を受け取る >>
(action_tile "tate"
  (strcat
    "(setq tate (get_tile ¥"tate¥"))"
    "(mode_tile ¥"yoko¥" 2)"
  )
)
;<< edit_box 処理 yoko<B> に値を受け取る >>
(action_tile "yoko"
    "(setq yoko (get_tile ¥"yoko¥"))"
)
```

Edit_Box	mode の値 <value> が <3> のケース

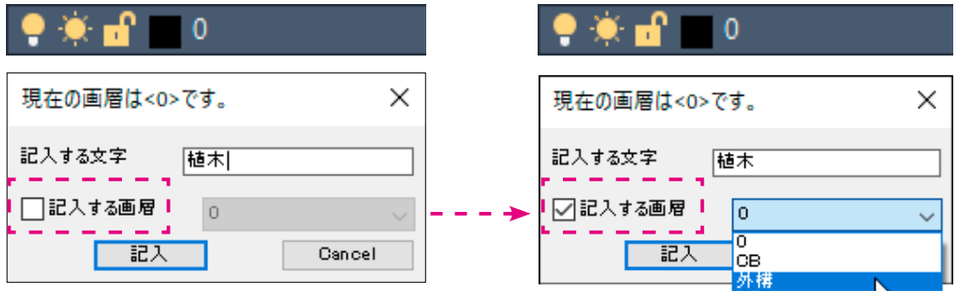


B に入力 <50> して、エンターキー（改行）を押すと、B の値が選択された状態になります。

```
;<< edit_box 処理 tate<A> に値を受け取る >>
(action_tile "tate"
  (strcat
    "(setq tate (get_tile ¥"tate¥"))"
    "(mode_tile ¥"yoko¥" 2)"
  )
)
;<< edit_box 処理 yoko<B> に値を受け取る >>
(action_tile "yoko"
    "(setq yoko (get_tile ¥"yoko¥"))"
    "(mode_tile ¥"yoko¥" 3)"
)
```


Toggle_Button	mode の値 <value> が <0> と <1> のケース

[記入する画層] にチェックを付けると、[画層一覧] の Popup_List が開きます。



タイルの <On> と <Off> を反転させるには、以下の 2 通りがあります。

① (action_tile "chk1" "(mode_tile ¥"layer_sel¥" (- 1 (atoi (get_tile ¥"chk1¥"))))")
減算処理により、popup_tile "layer_sel" の mode を反転しています。

☞P2-133 の⑤を参考

② toggle_button "chk1" が <On> か <Off> された時の処理をサブ関数に記述する方法です。

```
;toggle "chk1" が変更された時、サブ関数 (chk_toggle) を起動する
(action_tile "chk1" "(chk_toggle)")
; サブ関数を以下に記述
(defun chk_toggle()
;toggle_button "chk1" の値 <"0"> か <"1"> を整数に変換して変数 [chk] にセット
  (setq chk (atoi (get_tile "chk1")))
    (if (= chk 1)                    ; もし、[chk] の値が <1> のときは、次の 1 行目を実行
      (mode_tile "layer_sel" 0)      ;popup_tile "layer_sel" の mode を <0>（表示）にする
      (mode_tile "layer_sel" 1)      ;[chk] の値が <1> でなければ、"layer_sel" の mode を <1>（非表示）
    );if                             ; にする
  );end
```

Point!

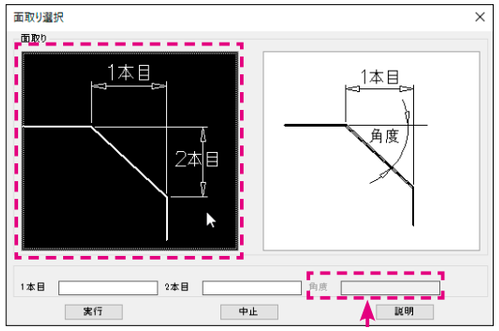
toggle_button は action された時、その値が <"1"> か <"0"> か判りません。
（radio_button は action された時は、必ず <"1"> です。）
そのため、その値が <"1"> か <"0"> かを判断する必要があります。

以下のように変数 [a] に値をセットし、その値が <"1"> か <"0"> かによって、プログラムを分岐します。

```
(setq  a (atoi (get_tile  "chk1")))          ;"chk1" は <"1"> か <"0"> の文字なので、整数に変換します。
(if  ( = a  1) (a が <1> の時のプログラム )
               (a が <0> の時のプログラム )
);if
```

Icon_Image	mode の値 <value> が <0> と <1> のケース（第 2 部 3 章 5 節より）

Step1 ー左側の図を選択すると、[角度] のタイルが非表示になり、角度は入力できません。

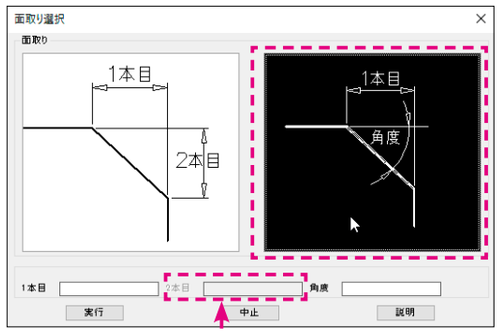


左側のイメージのダイアログの記述
```
:icon_image
{
  label = "";
  key = "image01";
  width = 30;
  height =15;
}
```

選択できません！

```
;<< image01 処理 変数 check=1 >>
 (action_tile "image01"          ;icon_image "image01" を選択したときの動作を記述する
   (strcat
     "(setq check 1)"            ; 変数 [check] の値を <1> にセット（後でプログラムの分岐に使う）
     "(mode_tile ¥"kaku¥" 1)"    ;edit_box "kaku" を非アクティブ（入力不可能）にする
     "(mode_tile ¥"sen2¥" 0)"    ;edit_box "sen2" をアクティブ（入力可能）にする
   )
 )
```

Step2 ー右側の図を選択すると、[2 本目] のタイルが非表示になり、2 本目の入力はできません。



右側のイメージのダイアログの記述
```
:icon_image
{
  label = "";
  key = "image02";
  width = 30;
  height =15;
}
```

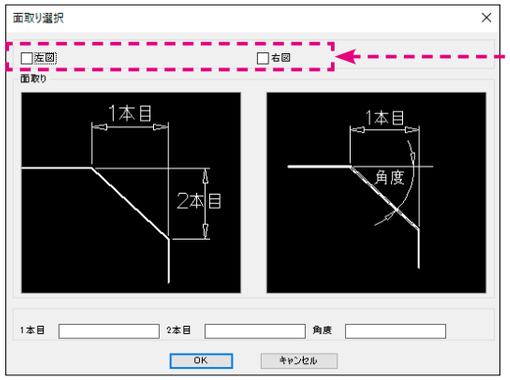
選択できません！

```
;<< image02 処理 変数 check=2 >>
 (action_tile "image02"          ;icon_image "image02" を選択したときの動作を記述する
   (strcat
     "(setq check 2)"            ; 変数 [check] の値を <2> にセット（後でプログラムの分岐に使う）
     "(mode_tile ¥"sen2¥" 1)"    ;edit_box "sen2" を非アクティブ（入力不可能）にする
     "(mode_tile ¥"kaku¥" 0)"    ;edit_box "kaku" をアクティブ（入力可能）にする
   )
 )
```


Icon_Image	mode の値 <value> が <4> のケース

Step1 ープログラムを起動した時は、両方のイメージの背景はオフ（黒色）です。

左図と右図の背景色は黒色 < オフ >



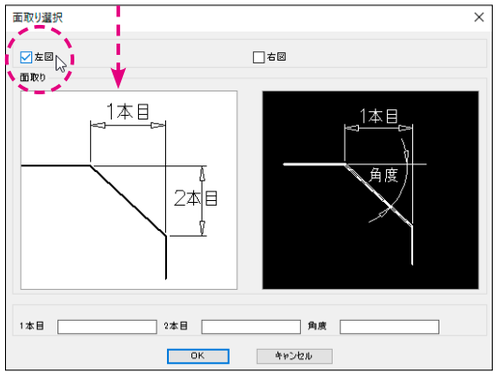
2 つの toggle_button の記述
```
:toggle
{
  label = " 左図 ";
  key = "hidari";
}
:toggle
{
  label = " 右図 ";
  key = "migi";
}
```

起動時は、両方のイメージの背景を黒 < オフ > にセットしておきます。

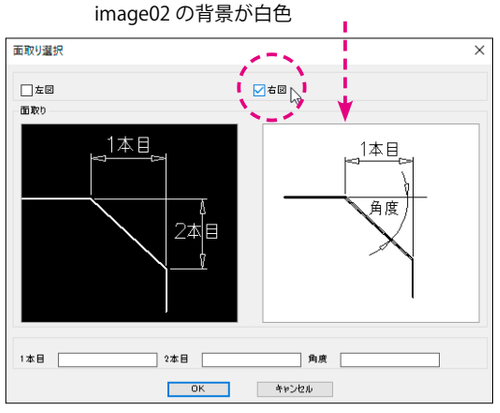
```
(mode_tile "image01" 4)
(mode_tile "image02" 4)
```

Step2 ー toggle の [左図] を選択（チェックする）と左のイメージ (image01) のハイライトがオン（背景が白色）になります。
同様に、toggle の [右図] を選択（チェックする）と右のイメージ (image02) のハイライトがオン（背景が白色）になります。

チェックボックス (hidari) をオンにすると image01 の背景が白色



チェックボックス (migi) をオンにすると image02 の背景が白色



<< 左の toggle をチェックした時 (on) >>

```
(action_tile "hidari" "(mode_tile ¥"image01¥" 4)")
```

<< 右の toggle をチェックした時 (on) >>

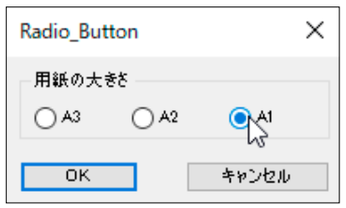
```
(action_tile "migi" "(mode_tile ¥"image02¥" 4)")
```

❹ Action_Tile

目 的	ダイアログ ボックスで特定のタイルを選択したときのアクションを割り当てます。
書 式	(action_tile "key" " 処理内容 ")

値を変数にセットする　(action_tile "A3" "(setq yousi 1)") のケース

```
:radio_button
{
  label = "A3";
  key = "A3";
}
:radio_button
{
  label = "A2";
  key = "A2";
}
:radio_button
{
  label = "A1";
  key = "A1";
}
```



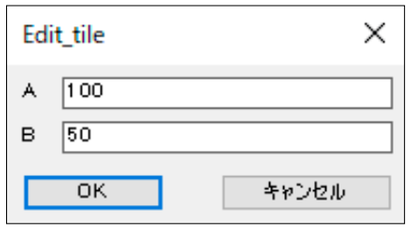

```
(action_tile "A3" "(setq yousi 1)")
(action_tile "A2" "(setq yousi 2)")
(action_tile "A1" "(setq yousi 4)")
```

変数にセットした値は条件分岐に使用します。

```
(cond
   ((= yousi 1) ( プログラム 1 を実行 ))
   ((= yousi 2) ( プログラム 2 を実行 ))
   ((= yousi 4) ( プログラム 3 を実行 ))
)
```

値を get_tile で取得する　(action_tile "tate" "(setq tate (get_tile ¥"tate¥"))") のケース

```
:edit_box
{
  label = "A";
  key = "tate";
}
:edit_box
{
  label = "B";
  key = "yoko";
}
```




```
(action_tile "tate" "(setq tate1 (get_tile ¥"tate¥"))")
(action_tile "yoko" "(setq yoko1 (get_tile ¥"yoko¥"))")
```

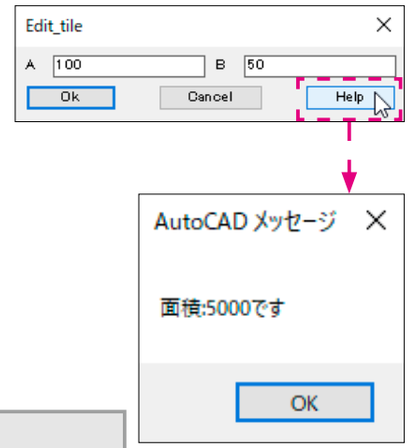
get_tile 関数でタイルから文字列の値を受け取る時の注意

タイル内の値は文字ですから、その値を文字以外で利用する場合は変換します。
（タイルの key は <tate>、受け取る値は <"3.14">）

```
値を文字で使用する場合・・・"(setq a (get_tile ¥"tate¥"))          ;a = "3.14"（そのまま）
    整数で使用する場合・・・"(setq a (atoi (get_tile ¥"tate¥")))"   ;a = 3
    実数で使用する場合・・・"(setq a (distof (get_tile ¥"tate¥")))" ;a = 3.14
```


サブ関数を起動する　(action_tile "help" "(GetTileVal)") のケース

```
: button
{
   label = "Cancel";
   is_cancel = true;
   key = "Cancel";
   fixed_width = true;
}
: button
{
   label = "Help";
   key = "help";
   fixed_width = true;
}
```

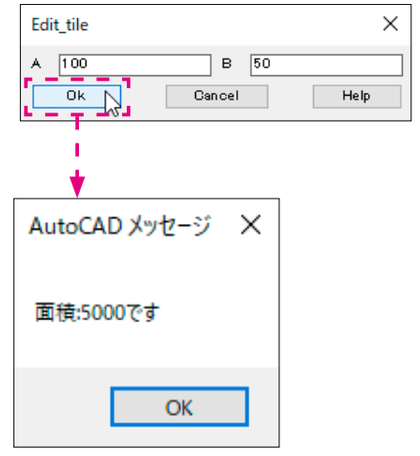



```
;<< help に内部サブ関数を与える >>
 (action_tile "help" "(GetTileVal)")
;help ボタンを押した時、サブ関数 (GetTileVal)") を起動する。
 (defun GetTileVal ()
  ;<< 面積の数値を変数 [goukei] にセットする >>
   (setq goukei (* (atof yoko) (atof tate)))
   (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
   (alert goukei_moji)
 )
```

メッセージを表示した後も、ダイアログは閉じない。

サブ関数を起動して終了する　(action_tile "accept" "(GetTileVal)(done_dialog)") のケース

```
;<< タイルの数値をゲットする（内部サブ関数）>>
 (defun GetTileVal ()
  ;<< 面積の数値を変数 [goukei] にセットする >>
   (setq goukei (* (atof yoko) (atof tate)))
   (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
   (alert goukei_moji)
 )
;<< [OK] ボタンと [Cancel] ボタンの動作を記述する >>
; サブ関数を起動して終了する
(action_tile "accept" "(GetTileVal)(done_dialog)")
; 何もしないで終了する
(action_tile "cancel" "(done_dialog)")
```



[Help] ボタンにサブ関数を割り当てたときは、[Help] を終了してもダイアログは閉じませんが、[OK] ボタンにサブ関数を割り当てたときは、ダイアログは終了します。

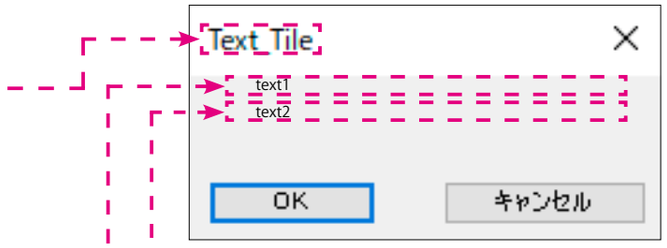
メッセージを表示した後、ダイアログは閉じる。

第 3 節　Text_Tile（線分と円の数を表示する）

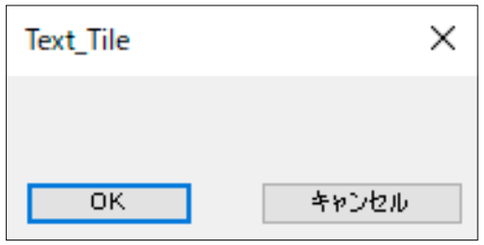
使用するダイアログ	S2_Text.dcl

```
S2_text : dialog
{
 label = "Text_Tile";
 key = "top_label";
 : column
 {
   : text/* 線分の数 */
   {
    label = "";
    key = "text1";
    width = 30;
   }
   : text/* 円の数 */
   {
    label = "";
    key = "text2";
    width = 30;
   }
 }
   spacer;
   ok_cancel;
}//end
```

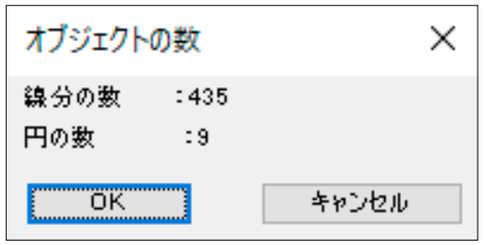
作成時の状態



起動時の状態



プログラム実行時の状態



Point!

①ダイアログのタイトル名は AutoLISP から変更できます。
タイトルの key（上記の場合は ["top_label"]）をターゲットに変更します。
→ (set_tile "top_label" " オブジェクトの数 ")

②ダイアログ内の各タイルの情報（数値や文字）も、その key をターゲットに取得や変更ができます。
・"text1" に文字列をセットする場合　→　(set_tile "text1" " 線分の数 ")
・"text2" の文字列を取得する場合　→　(setq a (get_tile "text2"))

使用する LISP	S2_Text.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun c:S2_text ( / lin1 num_lin cir1 num_cir dcl_id dialog-box)
; 線分の個数
  (setq lin1 (ssget "x" '((0 . "LINE"))))          ; 図面にある線分を全て取得する
  (setq num_lin (sslength lin1))                   ; 取得した線分の数をカウントする
; 円の個数                                                              → Step1
  (setq cir1 (ssget "x" '((0 . "CIRCLE"))))        ; 図面にある円を全て取得する
  (setq num_cir (sslength cir1))                   ; 取得した円の数をカウントする

; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_text.dcl"))        ;S2_text.dcl をメモリーに読み込む
  (new_dialog "S2_text" dcl_id)                    ;S2_text ダイアログを初期化する   → Step2

Ⓢ (set_tile "top_label" " オブジェクトの数 ")      ; ダイアログのタイトルを変更する   → Step3
; タイルに 2 つの文字を連結してセットする（タイルの値は文字列なので整数を文字に変換する）
Ⓢ (set_tile "text1" (strcat " 線分の数　:" (itoa num_lin)))  ; 線分の数を文字に変換
Ⓢ (set_tile "text2" (strcat " 円の数　:" (itoa num_cir)))    ; 円の数を文字に変換   → Step4

; ダイアログを終了する
  (setq dialog-box (start_dialog))                 ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                           ; ダイアログをメモリーから解放する  → Step5

(princ)
);end
```

Step1 ー図面内の線分と円の数を取得して、変数にセットします。
　　ダイアログを表示する前に計算を済ませておきます。

Step2 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step3 ーダイアログのタイトルを変更します。
　　（タイトルに "key" を指定していると可能です。）

Step4 ー "text1" と "text2" のタイルに計算結果の文字列を代入します。
　　[strcat] は、文字列を連結する関数です。
　　[itoa] は、整数を文字列に変換する関数です。

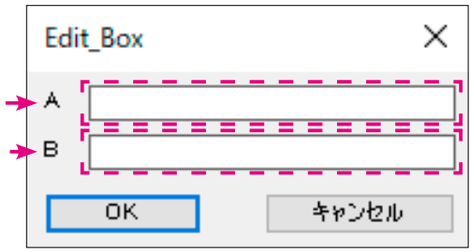
Step5 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

第 4 節　Edit_Box（数値入力を行う）

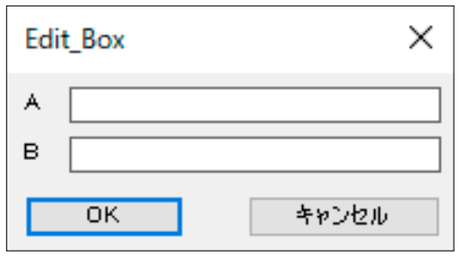
使用するダイアログ	S2_Edit.dcl

```
S2_edit :dialog
{
  label = "Edit_Box";

    :edit_box
    {
     label = "A";
     key = "tate";
     width = "";
     value = "";
    }
    :edit_box
    {
     label = "B";
     key = "yoko";
     width = "";
     value = "";
    }
 /* OK or CANCEL */
  spacer;
  ok_cancel;
}//end
```

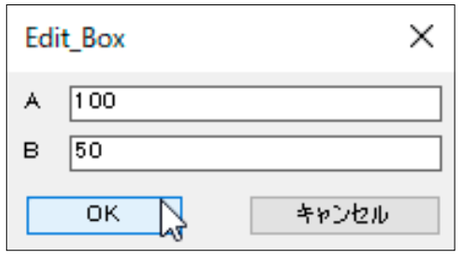
作成時の状態



起動時の状態



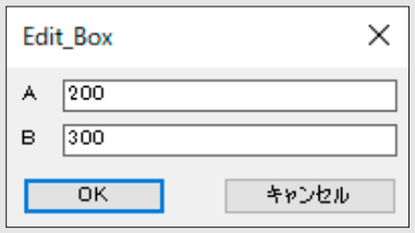
プログラム実行時の状態



Point!

```
{
    label = "A";
    key = "tate";
    width = "";
    value = "200";
}
{
    label = "B";
    key = "yoko";
    width = "";
    value = "300";
}
```

よく使う値を既定値にしておくには、ダイアログの中で記述しておきます。



[OK] ボタンを押した時に、この box の数値を取得するコールバック関数が必要です。

使用する LISP	S2_Edit.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun c:S2_edit( / dialog-box dcl_id)
; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_edit.dcl"))      ;S2_edit.dcl をメモリーに読み込む
  (new_dialog "S2_edit" dcl_id)                  ;S2_edit ダイアログを初期化する
                                                                          → Step1
;tate の初期設定
; ダイアログでセットした数値以外の数値をセットする場合
Ⓢ (set_tile "tate" "100")                        ;<"100"> をセットする
; yoko の初期設定
Ⓢ (set_tile "yoko" "50")                         ;<"50"> をセットする
                                                                          → Step2
; edit_box 処理 tate に値を受け取る
;mode_tile を <2> にすると、エンターキーを押した時に、カーソルが次のタイルに移動する
Ⓐ (action_tile "tate"                            ;"tate" を action した時の動作を記述する
    (strcat                                      ; 複数の文字列を連結する
Ⓖ    "(setq tate (get_tile¥"tate¥"))"            ;"tate" の文字列を変数にセットする
Ⓜ    "(mode_tile ¥"yoko¥" 2)"                    ; カーソルを次のタイル "yoko" に移す
    )
  )                                                                       → Step3
; edit_box 処理 yoko に値を受け取る
Ⓐ (action_tile "yoko"                            ;"yoko" を action した時の動作を記述する
    (strcat                                      ; 複数の文字列を連結する（1 行でも可）
Ⓖ    "(setq yoko (get_tile ¥"yoko¥"))"           ;"yoko" の文字列を変数に代入する
    )
  )                                                                       → Step4
; ダイアログを終了する
  (setq chkdia (start_dialog))                   ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                         ; ダイアログをメモリーから解放する
                                                                          → Step5
);end
```

Step1 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step2 ー "tate" と "yoko" の初期値をそれぞれ <"100"> と <"50"> にセットします。

Step3 ー "tate" のタイルに入力を受け付けます。エンターを押すと "yoko" のタイルにカーソルが移動します。

Step4 ー "yoko" のタイルに入力を受け付けます。

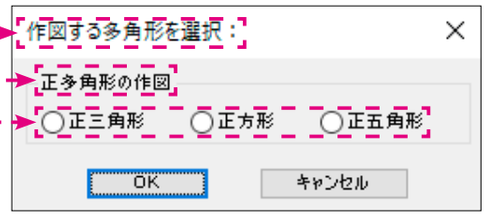
Step5 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

第 5 節　Radio_Button（ポリゴンを選択する）

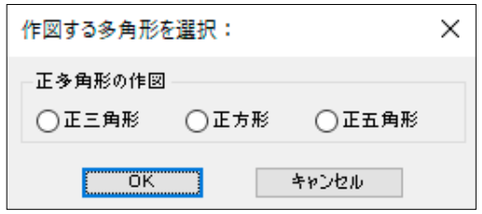
使用するダイアログ	S2_Radio.dcl

```
S2_radio : dialog
{
  label = "Radio_Button";
  key   = "top_label";
  :boxed_radio_row
  {
    label = " 正多角形の作図 ";
    key = "sel_poly";
      :radio_button
      {
        label = " 正三角形 ";
        key = "sankaku";
      }
      :radio_button
      {
        label = " 正方形 ";
        key = "sikaku";
      }
      :radio_button
      {
        label = " 正五角形 ";
        key = "gokaku";
      }
  }
  spacer;
  /* OK or CANCEL */
  ok_cancel;
}//end
```

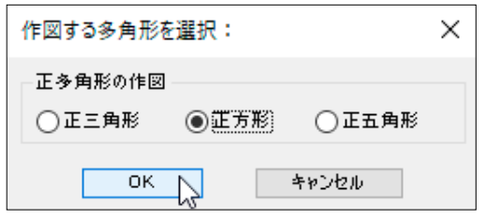
作成時の状態



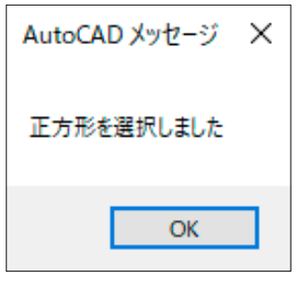
起動時の状態



プログラム実行時の状態



[OK] ボタンを押すと、メッセージを表示します



Point!

- radio_box の [key　= "sel_poly";] が取得する値は、<"sankaku"> か <"sikaku"> か <"gokaku"> です。
- 各 radio_button（"sankaku"、"sikaku"、"gokaku"）が取得する値は、<"0"> か <"1"> です。

使用する LISP	S2_Radio.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun c:S2_radio( / dialog_box dcl_id poly)
; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_radio.dcl"))      ;S2_radio.dcl をメモリーに読み込む
  (new_dialog "S2_radio" dcl_id)                  ;S2_radio ダイアログを初期化する
;set_tile でダイアログのタイトルを変更
Ⓢ (set_tile "top_label" " 作図する多角形を選択：") ; ダイアログのタイトルを変更する
;radio_button が押された時の動作を記述する
Ⓐ (action_tile "sankaku" "(setq poly 1)")       ; 変数 [poly] に <1> をセットする
Ⓐ (action_tile "sikaku" "(setq poly 2)")        ; 変数 [poly] に <2> をセットする
Ⓐ (action_tile "gokaku" "(setq poly 4)")        ; 変数 [poly] に <4> をセットする
   (if (= poly nil)                             ; どれも選択されなかったら
      (setq poly 2));if                         ;<2> をセットする
; ダイアログを終了する
  (setq dialog-box (start_dialog))              ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                        ; ダイアログをメモリーから解放する
;[OK] ボタンが押された時の処理を記述する
  (if (= (getvar "DIASTAT") 1)                  ;[OK] ボタンが押された場合の処理
    (cond
     ((= poly 1)(alert " 正三角形を選択しました "))  ;[poly] が <1> の時のメッセージを表示
     ((= poly 2)(alert " 正方形を選択しました "))    ;[poly] が <2> の時のメッセージを表示
     ((= poly 4)(alert " 正五角形を選択しました "))  ;[poly] が <4> の時のメッセージを表示
    );cond
  );if
(princ)
);end
```

→ Step1　→ Step2　→ Step3　→ Step4　→ Step5

Step1 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step2 ーダイアログのタイトルを変更します。

Step3 ー radio_button が押された時の処理を記述します。
"sankaku" → <1>、"sikaku" → <2>、"gokaku" → <4>

Step4 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

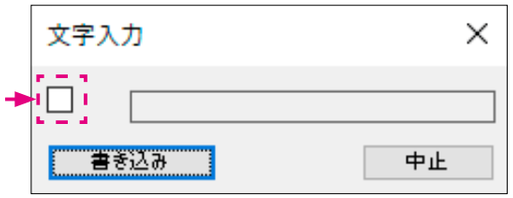
Step5 ー [Ok] ボタンが押された時、変数 [poly] の値によって alert 関数で表示する文字列を分岐処理します。

第 6 節　Toggle_Button（文字入力を可能にする）

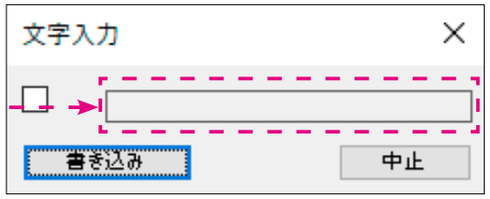
使用するダイアログ	S2_Toggle.dcl

```
S2_toggle : dialog
{
  label = " 文字入力 ";
  spacer;
  :row
  {
   :toggle
   {
    label = "";
    key = "chk1";
   }
   :edit_box
   {
    key = "text1";
    width = 30;
    value = "";
   }
  }
  spacer;
  /* OK or CANCEL */
  : row
  {
   : button
   {
    label = " 書き込み ";
    is_default = true;
    key = "accept";
    width = 8;
    fixed_width = true;
   }
   : button
   {
    label = " 中止 ";
    is_cancel = true;
    key = "Cancel";
    width = 8;
    fixed_width = true;
   }
  }
}//end
```

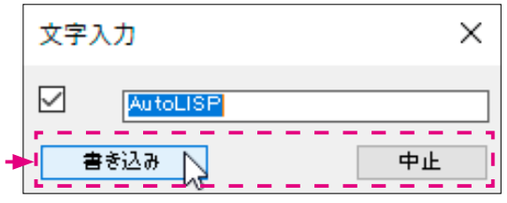
作成時の状態



起動時の状態（書き込みは不可）



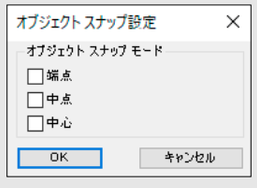
チェックをオン（書き込みが可能になる）



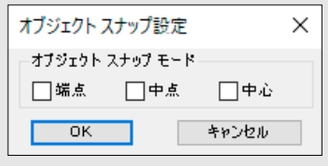
Point!

toggle_button のラベルは右側に表示されるので、一般的に縦型 (boxed_column) に配置します。
しかし、ラベルの文字が少ないためにバランスが悪くなる場合は、横型 (boxed_row) にします。

:boxed_column



:boxed_row



使用する LISP	S2_Toggle.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun c:S2_toggle( / dialog_box dcl_id)
; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_toggle.dcl"))      ;S2_toggle.dcl をメモリーに読み込む
  (new_dialog "S2_toggle" dcl_id)                  ;S2_toggle ダイアログを初期化する
                                                                              → Step1
;text_tile[text1] を使用禁止にする（toggle_button でコントロールするため）
Ⓜ (mode_tile "text1" 1)                            ;"text1" タイルの mode を <1> にセット  → Step2
;toggle_button[chk1] がチェックされた時の値 (<"0"> か <"1">) で "text1" タイルの mode を変更
Ⓐ
Ⓜ (action_tile "chk1" "(mode_tile¥"text1¥" (- 1 (atoi (get_tile ¥"chk1¥"))))")  → Step3
Ⓖ
;[OK] ボタンが押された時、このサブ関数を起動して、"text1" タイルにある値を取得する
    (defun SaveTileVal()                           ; コールバック関数を起動する
     (setq text1 (get_tile "text1"))               ;"text1" タイルの値を変数 [text1] にセット
    )                                                                         → Step4
  ;[OK] ボタンと [Cancel] ボタンの動作を記述する
Ⓐ (action_tile "accept"  "(SaveTileVal)(done_dialog 1)"); サブ関数に移る
Ⓐ (action_tile "cancel"  "(done_dialog 0)")        ; ダイアログの終了状態を <0> にセット  → Step5
; ダイアログを終了する
  (setq chkdia (start_dialog))                     ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                           ; ダイアログをメモリーから解放する      → Step6

    (if(= chkdia 1)                                ;[OK] ボタンが押された時の動作を記述
        (command "text" pause pause pause text1) ; 文字を書き入れる                    → Step7
    );if
);end
```

Step1 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step2 ー text_tile[text1] を使用禁止にします。（初期値）

Step3 ー (- 1 (atoi (get_tile ¥"chk1¥"))) → "chk1" が <"0"> であれば、"text1" の mode は <1>、<"1"> であれば、"text1" の mode は <0>。これによって、"text1" の mode は反転します。

Step4 ー [OK] ボタンが押された時、このサブ関数で "text1" の値を取得します。

Step5 ー [OK] ボタンが押されると、サブ関数を起動し、同時にダイアログの終了状態をセットします。

Step6 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

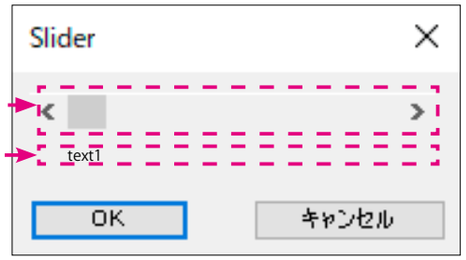
Step7 ー [OK] ボタンが押された時、"text1" の文字を図面に書き入れます。

第 7 節　Slider（スライダーの数値を取得する）

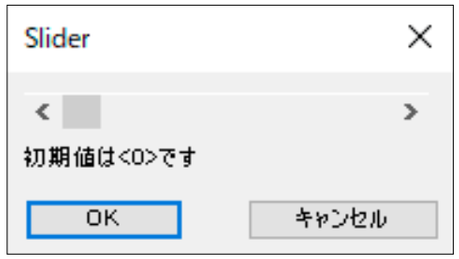
使用するダイアログ	S2_Slider.dcl

```
S2_slider : dialog
{
 label = "Slider";
 spacer;
 :slider
 {
  key = "slider1";
  min_value = 0;
  max_value = 100;
  small_increment = 10;
  big_increment = 20;
  width = 30;
  fixed_width = true;
  is_tab_stop = false;
 }
 :text
 {
  key = "text1";
  width = 20;
 }
 spacer;
 /* OK or CANCEL */
 ok_cancel;
}//end
```

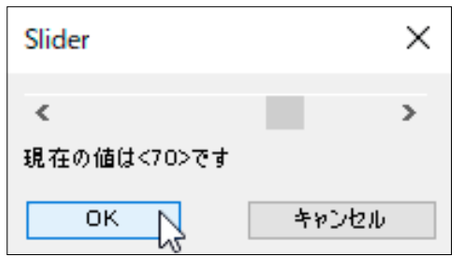
作成時の状態



起動時の状態

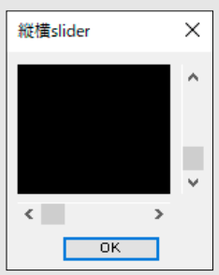


プログラム実行時の状態



Point!

slider には水平 (horizontal) と垂直 (vertical) の 2 種類がありますが、既定値は水平 (horizontal) です。
従って、水平にする場合には記述する必要はありませんが、垂直にする場合は、[layout = "vertical";] を付加します。



使用する LISP	S2_Slider.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun c:S2_slider( / dialog_box dcl_id p p1 pt0)
; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_slider.dcl"))      ;S2_slider.dcl をメモリーに読み込む
  (new_dialog "S2_slider" dcl_id)                  ;S2_slider ダイアログを初期化する
 ;slider1 の初期値を "0" にセットする。
Ⓢ (set_tile  "slider1" "0")                        ;slider が一番左端にセットされる
Ⓢ (set_tile  "text1" " 初期値は <0> です ")        ;"text1" に初期値の文字列をセット
 ;slider が動いた時に、その値を "text1" にセットして表示する
Ⓐ (action_tile "slider1"                           ;slider が動いた時
    (strcat                                        ; 文字列を結合する
Ⓖ    "(setq p (get_tile¥"slider1¥"))"              ;slider1 の値を変数 [p] にセット
     "(setq p1 (rtos (atof p)))"                   ;"text1" に代入するために文字に変換
     "(setq pt0 ¥" 現在の値は <¥")"                ; 表示用の前半の文字列をセット
Ⓢ    "(set_tile ¥"text1¥" (strcat pt0  p1 ¥"> です ¥"))" ;3 つの文字列を連結して "text1" に
    )                                              ; 表示する
  )
  ; ダイアログを終了する
  (setq dialog-box (start_dialog))                 ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                           ; ダイアログをメモリーから解放する

 (princ)
);end
```

Step1 ー ダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step2 ー "slider1" の初期値を、<"0"> にセットします。
"text1" に文字列 <" 初期値は <0> です "> をセットします。

Step3 ー "slider1" が動作したときの処理を記述します。
(rtos (atof p)) → p の値は実数値なので文字列に変換します。
もし p の値が <50> であれば、文字列は次のように表示されます。
" 現在の値は <50> です "

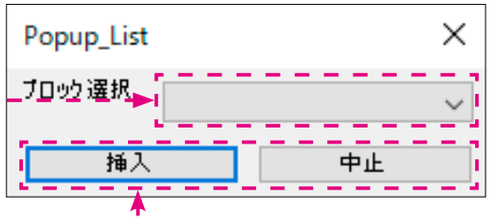
Step4 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

第 8 節　Popup_List（ブロックを挿入する）

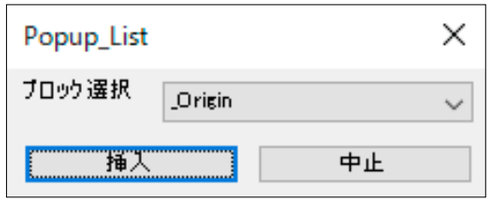
使用するダイアログ	S2_Popup.dcl

```
S2_popup :dialog
{
 label = "Popup_List";

 :popup_list
 {
  label = " ブロック選択 ";
  key = "list1";
  width = 35;
  value = "";
 }
 spacer;
 /* OK or CANCEL */
 : row
 {
  : button
  {
   label = " 挿入 ";
   is_default = true;
   key = "accept";
   width = 8;
   fixed_width = true;
  }
  : button
  {
   label = " 中止 ";
   is_cancel = true;
   key = "Cancel";
   width = 8;
   fixed_width = true;
  }
 }
}//end
```

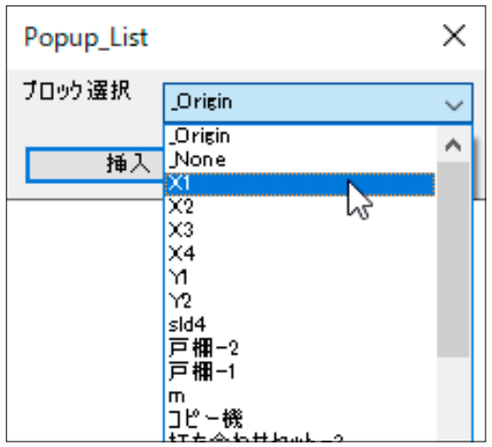
作成時の状態



起動時の状態



プログラム実行時の状態



Point!

リストのデータを list_box 内に追加するには、次の順番に沿って行います。

① (start_list)・・リストの処理を開始します。
② (add_list)・・リスト内に文字列を追加します。
③ (end_list)・・リスト処理の終了を通知します。

使用する LISP	S2_Popup.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun C:S2_popup( /  blkname blkdata blklst dcl_id sel_block chk1)
; 図面内にあるブロックを順番に読み込み、1 つのリストにします。
 (setq blkname (tblnext "BLOCK" T))          ; 最初のブロックを読み込む
 (while blkname                              ; blkname が nil を返すまで繰り返す
  (setq blkdata (cdr (assoc 2 blkname)))     ; blkname のブロック名を取得する
  (setq blklst (append blklst (list blkdata))) ; 取得したブロックをリストにしていく
  (setq blkname (tblnext "BLOCK"))           ; 次のブロック ( 無ければ nil を返す)
 );while                                                          → Step1
; ダイアログを呼び出す
 (setq dcl_id (load_dialog "S2_popup.dcl"))  ;S2_popup.dcl をメモリーに読み込む
 (new_dialog "S2_popup" dcl_id)              ;S2_popup ダイアログを初期化する   → Step2
;blklist のデータ を list_box"list1" に流し込んでいく
 (start_list "list1" 3)                      ;list_box に文字列を新規で流し込む
 (mapcar 'add_list blklst)                   ;blklst のデータを順番に追加していく
 (end_list)                                  ;list の終了を通知する             → Step3
;[OK] ボタンが押された時、このサブ関数を起動して、"list1" で選択された値（整数）を取得する
 (defun SaveTileVal()                        ; コールバック関数を起動する
Ⓖ (setq sel_block(nth (atoi(get_tile "list1")) blklst)) ; 選択された番号に符合したブロックを
 )                                           ; 変数 [sel_block] にセットする    → Step4
 ;[OK] ボタンと [Cancel] ボタンの動作を記述する
Ⓐ (action_tile "accept"  "(SaveTileVal)(done_dialog 1)") ; サブ関数に移る
Ⓐ (action_tile "cancel"  "(done_dialog 0)")  ; ダイアログの終了状態を <0> にセット → Step5
; ダイアログを終了する
 (setq chk1 (start_dialog))                  ; ユーザー入力を受け付ける
 (unload_dialog dcl_id)                      ; ダイアログをメモリーから解放する  → Step6
);end
```

Step1 ー図面内にあるブロックを全て取得して、1 つのリストにします。

Step2 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step3 ーリストにしたブロック名を list_box に新規で順番にセットしていきます。

Step4 ーリストで選択した項目は整数値ですから、その順番に符合したブロック名を取得します。

Step5 ー [OK] ボタンが押されると、サブ関数を起動し同時にダイアログの終了状態をセットします。

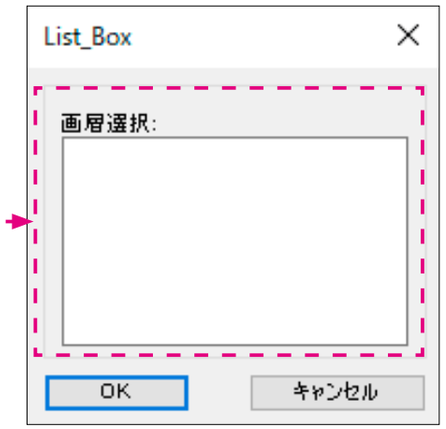
Step6 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

第 9 節　List_Box（画層を切り替える）

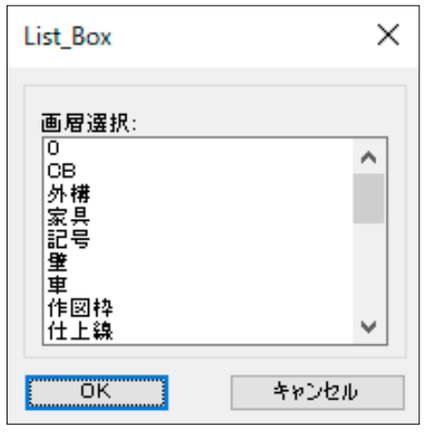
使用するダイアログ	S2_List.dcl

```
S2_list :dialog
{
  label = "List_Box";
  spacer;
    :list_box
     {
       label = " 画層選択 ";
       key = "list1";
       multiple_select = false;
       width = 20;
       height = 10;
     }
  /* OK or CANCEL */
  spacer;
  ok_cancel;
}//end
```

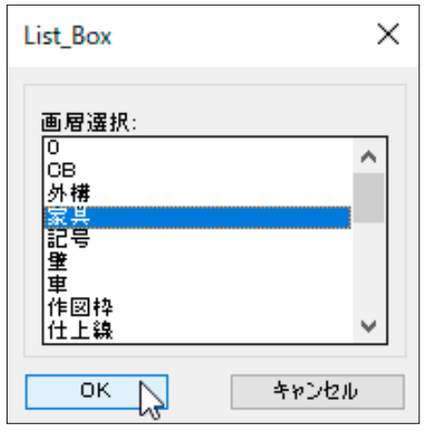
作成時の状態



起動時の状態

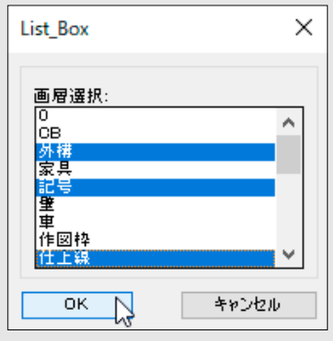


プログラム実行時の状態



Point!

List_Box には単一選択 (multiple_select = false;) と複数選択 (multiple_select = true;) の 2 種類がありますが、既定値は単一選択 (multiple_select = false;) です。
従って、単一選択にする場合には記述する必要はありませんが、複数選択にする場合は、[multiple_select = true;] に変更または付加します。



使用する LISP	S2_List.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun C:S2_list( / layname laydata laylst dcl_id sel_Layer chk1)
; 図面内にある画層名を順番に読み込み、1 つのリストにします。
  (setq layname (tblnext "LAYER" T))          ; 最初の画層名を読み込む
  (while layname                              ; layname が nil を返すまで繰り返す
  (setq laydata (cdr (assoc 2 layname)))      ; layname の画層名を取得する
  (setq laylst (append  laylist (list  laydata)))  ; 取得した画層名をリストにしていく
  (setq layname (tblnext "LAYER")))           ; 次の画層 ( 無ければ nil を返す )
; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_list.dcl"))   ;S2_list.dcl をメモリーに読み込む
  (new_dialog "S2_list" dcl_id)               ;S2_list ダイアログを初期化する
;laylst のデータ を list_box"list1" に流し込んでいく
  (start_list "list1" 3)                      ;list_box に文字列を新規で流し込む
  (mapcar 'add_list laylst)                   ;laylst のデータを順番に追加していく
  (end_list)                                  ;list の終了を通知する
;[OK] ボタンが押された時、このサブ関数を起動して、"list1" で選択された値（整数）を取得する
  (defun GetTileVal()                         ; コールバック関数を起動する
Ⓖ (setq sel_Layer(nth (atoi(get_tile "list1")) laylst)) ; 選択された番号に符合した画層名を
  )                                           ; 変数 [sel_layer] にセットする
  ;[OK] ボタンと [Cancel] ボタンの動作を記述する
Ⓐ (action_tile "accept"  "(GetTileVal)(done_dialog 1)")  ; サブ関数に移る
Ⓐ (action_tile "cancel"  "(done_dialog 0)")  ; ダイアログの終了状態を <0> にセット
  ; ダイアログを終了する
  (setq chk1 (start_dialog))                  ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                      ; ダイアログをメモリーから解放する
);end
```

→ Step1 → Step2 → Step3 → Step4 → Step5 → Step6

Step1 ー図面内にある画層名を全て取得して、1 つのリストにします。

Step2 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step3 ーリストにした画層名を list_box に新規で順番にセットしていきます。

Step4 ーリストで選択した項目は整数値ですから、その順番に符合した画層名を取得します。

Step5 ー [OK] ボタンが押されると、サブ関数を起動し同時にダイアログの終了状態をセットします。

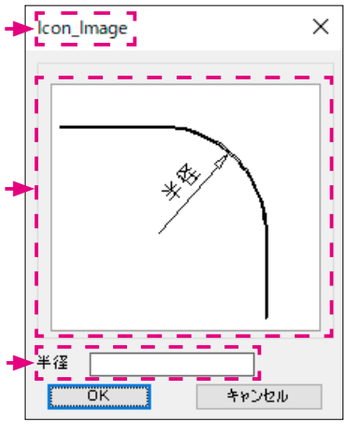
Step6 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

第 10 節　Icon_Image（半径を入力する）

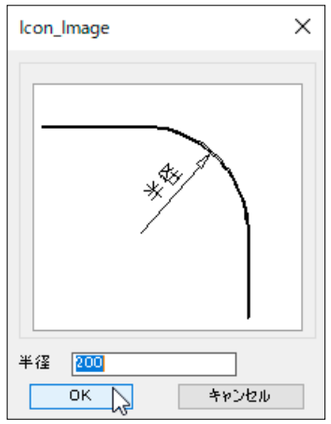
使用するダイアログ	S2_Image.dcl

```
S2_image : dialog
{
  label = "Icon_Image";
  key = "top_label";
    :boxed_row
    {
      label = "";
      :icon_image
      {
        label = "";
        key = "image01";
        width = 30;
        height =15;
      }
    }
    :row
    {
        :edit_box
        {
          label = " 半径 ";
          key = "han";
          width = 6;
          fixed_width = true;
          value = "";
        }
    }
  /* OK or CANCEL */
  ok_cancel;
}//end
```

作成時の状態



入力時の状態



イメージ タイル処理関数

関数名	説　明
(dimx_tile key)　(dimy_tile key)	タイルの寸法をダイアログ ボックス計測単位で取得します。
(start_image key)	ダイアログ ボックス タイルでのイメージの作成を開始します。
(fill_image x1 y1 wid hgt color)	現在アクティブなダイアログ ボックス イメージ タイルに、塗り潰された長方形を描きます。
(slide_image x1 y1 wid hgt sldname)	現在のアクティブなダイアログ ボックス イメージ タイルに、AutoCAD スライドを表示します。
(end_image)	アクティブなダイアログ ボックス イメージの作成を終了します。

使用する LISP	S2_Image.lsp

Ⓢ set_tile Ⓐ action_tile Ⓖ get_tile Ⓜ mode_tile

```
(defun c:S2_image( / dialog_box dcl_id)
 ; ダイアログを呼び出す
  (setq dcl_id (load_dialog "S2_image.dcl"))   ;S2_image.dcl をメモリーに読み込む
  (new_dialog "S2_image" dcl_id)               ;S2_image ダイアログを初期化する
 ; イメージ（slide ファイル）を表示する処理
  (setq x1 0) (setq y1 0)                      ;slide の左上の座標をセット
  (setq x2 (dimx_tile "image01")               ;slide の X 方向（右）の距離をセット
        y2 (dimy_tile "image01")               ;slide の Y 方向（下）の距離をセット
  )
 ;slide ファイルを "iamge01" に表示する
  (start_image "image01")                      ;"image01" に slide の表示を開始する
  (fill_image x1 y1 x2 y2 -2)                  ; 背景色を図面の背景色と同じにする
  (slide_image x1 y1 x2 y2 "S_Edit3-F")        ; 表示する slide ファイルを指定する
  (end_image)                                  ;slide のセットを終了する
 ; ダイアログのタイトルを変更する
Ⓢ (set_tile "top_label" " フィレット ")        ; タイトルを [ フィレット ] に変更する
 ; edit_box "han" の値を変数 [han1] にセットする
Ⓐ (action_tile "han"                           ;"han" に値が入力された時の処理
Ⓖ       "(setq han1 (get_tile¥"han¥"))")       ;"han" の値を変数 [han1] にセットする
 ; ダイアログを終了する
  (setq dialog-box (start_dialog))             ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                       ; ダイアログをメモリーから解放する
);end
```

→ Step1　→ Step2　→ Step3　→ Step4　→ Step5　→ Step6

Step1 ーダイアログを読み込み、ダイアログ内のタイルを初期化します。

Step2 ー slide ファイルを表示する範囲を [左上→右下] の順番で指定します。

Step3 ー slide ファイルを指定した XY の範囲に表示します。

Step4 ーダイアログのタイトルを [フィレット] に変更します。

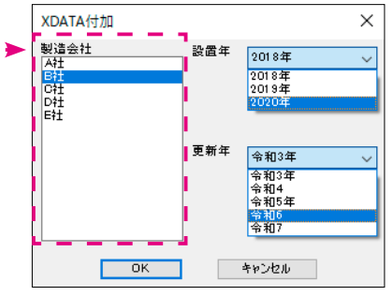
Step5 ー "han" に入力された値を変数 [han1] にセットします。（フィレットの半径）

Step6 ー [Ok] または [Cancel] ボタンが押されると、ダイアログを終了します。

第 11 節　リストの作り方（Popup_List、List_Box 共通）

1 AutoLISP の最初にリストを記述する

```
; リストを作成し、変数 [list1_data] にセットする
(setq list1_data (list "A 社 " "B 社 " "C 社 " "D 社 " "E 社 "))
; list_box"list1" を新規でスタート
 (start_list "list1" 3)
; リスト "list1" 内に <list1_data> の内容を流し込む
 (mapcar 'add_list list1_data)
; リストの配置を終了
 (end_list)
```

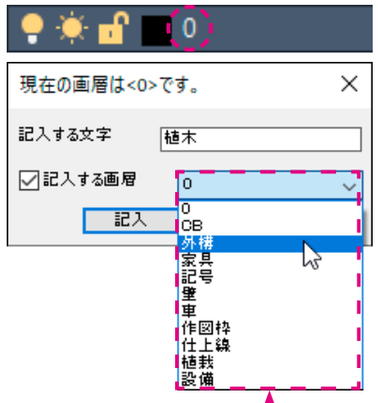


リスト ボックスとポップアップ リスト処理関数	
関数名	説　明
(start_list key [operation])	リスト ボックスまたはポップアップ リストの処理を開始します。
(add_list string)	現在アクティブなダイアログ ボックス リスト内の文字列の追加や修正を行います。
(end_list)	現在アクティブなダイアログ ボックス リストの処理を終了します。

start_list の [operation] のコード	
値	説　明
1	選択したリスト項目の内容を変更します。
2	新しいリスト項目を追加します。
3	古いリストを削除して新しいリストを作成します (既定)。

2 図面内の全画層名を取得して、リストを作成する

```
; 最初の画層名を取得する
(setq layname (tblnext "LAYER" T))
; layname が nil を返すまで繰り返す
(while layname
; layname の画層名を取得する
(setq laydata (cdr (assoc 2 layname)))
; 取得した画層名をリストにしていく
(setq lst (append lst (list laydata)))
; 次の画層を取得する ( 無ければ nil を返す )
(setq layname (tblnext "LAYER")))
```



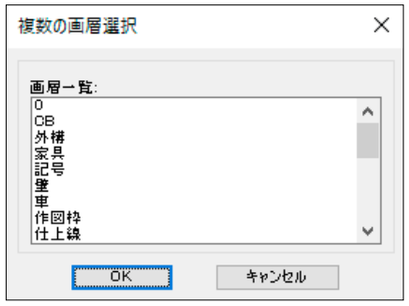

```
(start_list "layer_sel" 3)      ; list_box"layer_sel" を新規でスタート
(mapcar 'add_list lst)          ; list_box"layer_sel" 内に <lst> の内容を流し込む
(end_list)                      ; list_box"layer_sel" へのリスト追加を終了
```


3 外部のテキストファイルを読み込んで、リストを作成する

```
;<< テキスト ファイルを読み込みモードで開く >>
; ドライブを指定する場合は、<"C:¥¥autolisp¥¥out_layer.txt">
  (setq  in_layer  (open  "out_layer.txt"  "r"))
; テキストファイルから 1 行目を読み込む
  (setq  layer_name (read-line  in_layer))
; 読み込んだ画層名のリストを作成する
  (setq  lst (LIST layer_name))
; 画層名が無くなるまで読み込みを続ける
  (while (/= layer_name nil)
;1 行ずつ読み込む
    (setq  layer_name (read-line  in_layer))
; 読み込んだ画層名を 1 つのリストに連結していく
    (setq  lst (append lst (LIST layer_name)))
  );while
; ファイルを閉じる（必須です）
  (close  in_layer)
```

テキストファイルを作成しておきます。

テキストファイルから画層名をリストに表示




```
;<< リスト内に画層名を流し込む >>
  (start_list "layer_sel" 3)
; list_box"layer_sel" を新規でスタート
;<<lst から 1 つずつ取り出してリストに追加 >>
; カウントの初期値を <0> にセット
  (setq i 0)
; リストが空になるまで繰り返す
  (while (/= (nth i lst) nil)
; 取り出した画層名をリストに追加する
    (setq add_layer (nth i lst))
; "layer_sel" 内に <lst> の内容を流し込む
    (add_list add_layer)
; カウントを <1> プラスして、繰り返す
    (setq i (+ i 1))
  );while
; "layer_sel" へのリスト追加を終了
  (end_list)
```

Point!

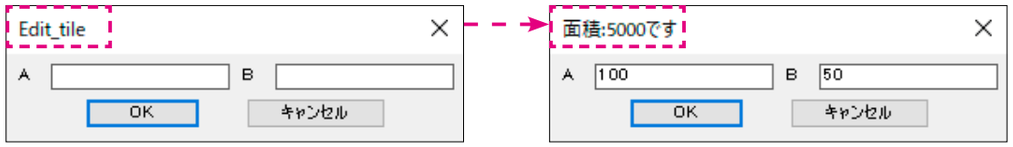
左のプログラムは下記のように
簡単に記述できます。

```
; list_box"layer_sel" を新規でスタート
(start_list "layer_sel" 3)
 ; list_box"layer_sel" 内に <lst> の内容を流し込む
(mapcar 'add_list lst)
; list_box"layer_sel" へのリスト追加を終了
(end_list)
```

第 12 節 コールバック関数

1 Action 式コールバック (Edit_Box に記述する例)

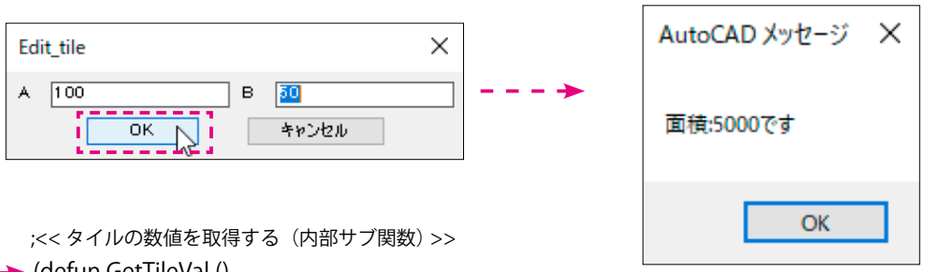
edit_box に action 式を記述します。（edit_box の数値をダイアログのラベルにセットします。）




```
(action_tile "tate"
  (strcat                                   ; 以下の文字列を結合する
    "(setq tate (get_tile ¥"tate¥"))"       ;edit_box "tate" の値を変数 [tate] にセット
    "(setq goukei (* (atof tate) (atof yoko)))"   ;[tate] と [yoko] を積算して変数 [goukei] にセット
    "(setq goukei_moji (strcat ¥" 面積 :¥" (rtos goukei) ¥" です ¥"))"
    "(set_tile¥"goukei¥" goukei_moji)"      ; ダイアログのタイトルのキー "goukei" に [goukei_moji]
  )                                         ; を代入
)
```

2 Action 式コールバック (Button に割り当てる例)

[OK] ボタンに action 式を記述します。（[OK] ボタンを押した時に、コールバック関数を起動します。）




```
;<< タイルの数値を取得する（内部サブ関数）>>
(defun GetTileVal ()
;<< 面積の数値を変数 [goukei] にセットする >>
  (setq goukei (* (atof tate) (atof yoko)))      ;[tate] と [yoko] を積算して変数 [goukei] にセット
  (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
  (alert goukei_moji)                            ;[tate] と [yoko] を積算した結果を、alert 関数で表示
 )
;<<[OK] ボタンを押したときに、コールバック関数（サブ関数 <GetTileVal>）を起動する >>
(action_tile "accept" "(GetTileVal)(done_dialog)")   ;[OK] ボタンを押したときに、サブ関数を起動して終了
(action_tile "cancel" "(done_dialog)")               ;[Cancel] ボタンを押したときは、何もせずに終了
```


第 13 節 エラーを回避する

1 ファイルの呼び出し時のエラーチェック

①ダイアログの呼び出し時にエラーをチェックし、もし指定したダイアログが無い場合は
プログラムを終了します。

```
(setq dcl_id (load_dialog "S_Edit.dcl"))     ; DCL ファイルをロードします。
  (if (not (new_dialog "S_Edit" dcl_id))     ; ダイアログを初期化します。
      (exit)                                 ; ダイアログの初期化に失敗したときは終了します。
  );if
 );setq
```

②外部関数を呼び出すときは、先にロード関数で読み込んで使用します。

```
(action_tile "help"
  (strcat                                    ; 文字列を結合する
    "(load ¥"S_Toggle1¥")"                   ; 外部関数 [S_Toggle1.lsp] を読み込む
    "(C:S_Toggle1)"                          ;[S_Toggle1.lsp] を実行する
  )
)
```

2 データの型を変換する

① set_tile 関数でタイルに文字列をセットする時の注意

タイル内にセットする値は文字ですから、数値の場合は文字に変換してセットします。

（タイルの key は <text1>、セットする値は <a>）

```
変数 <a> が文字の場合・・・(set_tile "text1" a)              ; そのままセットする
        整数の場合・・・(set_tile "text1" "(itoa a)")        ; 整数を文字列に変換してセットする
        実数の場合・・・(set_tile "text1" "(rtos a)")        ; 実数を文字列に変換してセットする
        角度の場合・・・(set_tile "text1" "(angtos a)")      ; 角度を文字列に変換してセットする
```

② get_tile 関数でタイルから文字列の値を受け取る時の注意

タイル内の値は文字ですから、その値を文字以外で利用する場合は変換して受け取ります。

（タイルの key は <text1>、受け取る値は <"3.14">）

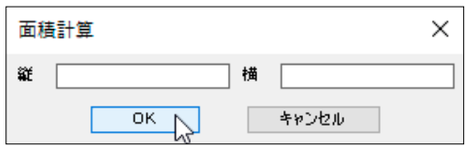
```
値を文字で使用する場合・・・(setq a (get_tile "text1"))          ;a = "3.14" （そのまま受け取る）
    整数で使用する場合・・・(setq a (atoi (get_tile "text1")))    ;a = 3
    実数で使用する場合・・・(setq a (distof (get_tile "text1")))  ;a = 3.14
    角度で使用する場合・・・(setq a (angtof (get_tile "text1")))  ;a = 0.0548033
```


タイルの機能とコールバック

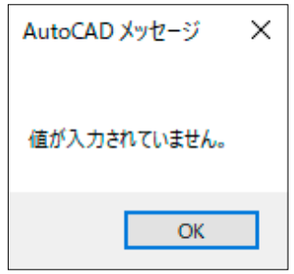
3 tile が空白の場合の処理

① edit_box（縦）の key は <tate>、（横）の key は <yoko>

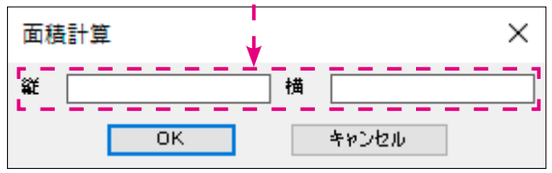
タイルに入力せずに、[OK] ボタンを押す



エラーメッセージを表示



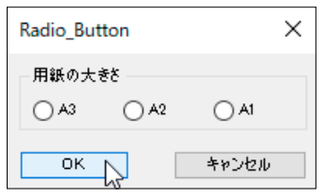

```
;<< タイルの数値をチェックする（値があれば終了、無ければエラーメッセージを表示する）>>
(defun CheckVal()
 (if (and (/= tate nil)(/= yoko nil))
     (done_dialog 1)                          ; 変数 [tate] と [yoko] に値が有れば、ダイアログを終了
     (progn                                   ; どちらかの値が無ければ、以下を実行する
       (alert "¥n 値が入力されていません。")    ;alert 関数を起動して原因を表示する
       (mode_tile  "tate"  2)                 ; カーソルのフォーカスを "tate" にセットする
     );progn
 );if
);end
```



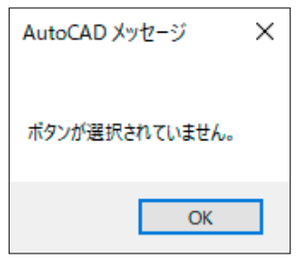

```
                 ・
            （中略）
                 ・
                 ・
;<<[OK] ボタンを押したときは、チェック関数を起動する。（ダイアログはチェック関数の中で閉じる。）>>
(action_tile "accept" "(CheckVal)")           ; チェック関数 (CheckVal) を実行する。終了しない。
(action_tile "cancel" "(done_dialog 0)")      ; 何も処理しないで閉じる
```

② toggle_row の key は <sel_yousi>

どのボタンも押さずに、[OK] ボタンを押す



エラーメッセージを表示




```
;<< ボタンが押されているかどうかチェックする >>
(defun CheckVal()
 (if (/= sel_yousi nil)  (done_dialog 1)      ;[sel_yousi] に値が有れば、ダイアログを終了
     (alert "¥n ボタンが選択されていません。")  ; 値が無ければ、エラーメッセージを表示する
 );if
);end
          （中略）
;<<[OK] ボタンを押したときは、チェック関数を起動する。（ダイアログはチェック関数の中で閉じる。）>>
(action_tile "accept" "(CheckVal)")           ; チェック関数 (CheckVal) を実行する。終了しない。
(action_tile "cancel" "(done_dialog 0)")      ; 何も処理しないで閉じる
```


第 3 章　タイルをコントロールする

ユーザーが AutoCAD と対話する手段としてダイアログは
視覚的に判りやすく、なじみやすい機能です。

この章では、DCL の組み立てから LISP のプログラミングを
通してユーザー インターフェースの構築を解説します。

第 1 節　OK_Cancel_Help ボタン

使用する LISP	Ok_Cancel2.lsp	使用する主な関数	getvar、itoa、rtos、atoi、atof
内部サブ関数	(GetTileVal)	外部サブ関数	(Ok_Cancel2_1)
使用する DCL	Ok_cancel2.dcl	使用するタイル	edit_box
外部テキスト	なし	スライドファイル	なし
このプログラムは、下図のダイアログを使用して、[OK] ボタンと [Help] ボタンの動作を確認します。 ダイアログに入力した数値や文字情報の取得方法を説明します。			

1　計算処理をサブ関数を使わずに行う。（次ページ 1 ）

計算処理や結果を表示する処理をサブ関数を使用せずに行います。

```
;<< 面積の数値を変数 [goukei] にセットする >>
(setq goukei (* (atof tate) (atof yoko))) ;"tate" と "yoko" の値を乗算する
(setq goukei_moji (strcat " 面積 :" (rtos goukei) " です ")) ; 結果を表示する文字列を作成 する
;<< ダイアログを終了する >>
 (setq dialog-box (start_dialog))
 (unload_dialog dcl_id)
```

2　計算処理を内部サブ関数か外部サブ関数で行う。（次ページ 2 3 ）

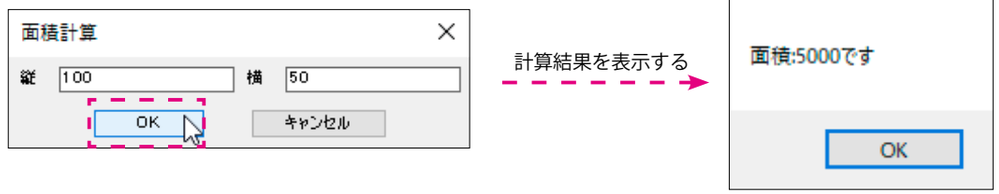
計算処理や結果を表示する処理をダイアログの内部で行うか（内部サブ関数）、プログラムの外（外部サブ関数）を使うかの 2 通りがあります。

```
(setq dcl_id (load_dialog "Ok_Cancel2"))   ; ダイアログをメモリーに読み込む
(new_dialog "Ok_Cancel2" dcl_id)           ; ダイアログを表示する

 ・・2 （内部サブ関数）

(setq dialog-box (start_dialog))           ; ユーザー入力を受け付ける
(unload_dialog dcl_id)                     ; ダイアログをメモリーから解放する
); プログラムの終了

 ・・3 （外部サブ関数）
```


1　サブ関数（コールバック関数）を使わずに、AutoCAD のメッセージボックスに表示する。

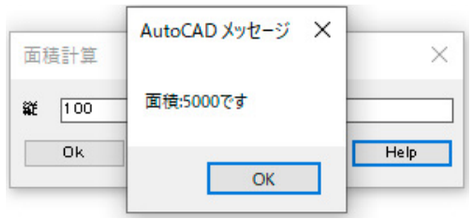
計算結果を AutoCAD のメッセージボックスへ表示する指示を、本プログラムの中で行います。
AutoCAD のメッセージボックスを表示した後、ダイアログを終了します。

[OK] ボタンを押すと、計算結果を表示して終了する。



2　内部サブ関数を使用して、AutoCAD のメッセージボックスに表示する。

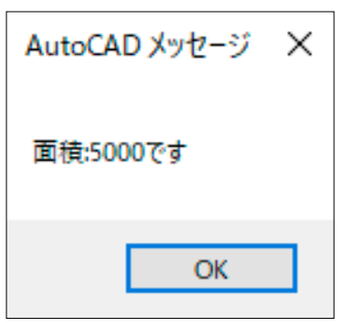
計算処理を本プログラム内部のサブ関数（コールバック関数）で処理して表示します。
[OK] ボタンで処理する場合と、[Help] キーで処理する場合があります。



[OK] ボタンを押した場合は、メッセージを閉じると
ダイアログも終了しますが、[Help] ボタンを使うときは、
メッセージを閉じると元のダイアログに戻ります。
他の処理（内部計算や別のプログラムを起動）した後でも、
ダイアログは残っています。

3　外部サブ関数を使用して、AutoCAD のメッセージボックスに表示する。

計算処理を本プログラム外部のサブ関数（コールバック関数）で処理して表示します。
分岐変数を使用して、使用するプログラムを選別指定することができます。

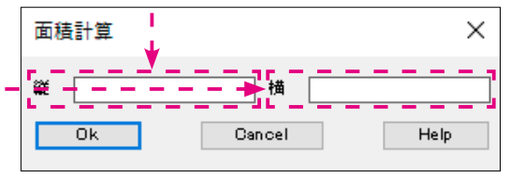


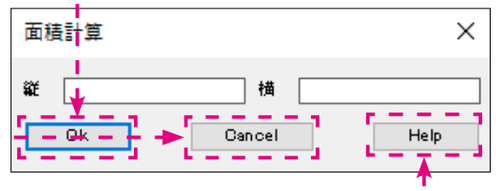
計算結果を AutoCAD のメッセージボックスへ表示する
指示を、本プログラムの外で行います。
プログラムが複雑な場合には、このようにした方がメン
テナンスに便利です。

しかし、本プログラムを終了して動作しますから、
再度、本プログラムに戻ることはできません。

使用するダイアログ	Ok_cancel2.dcl

```
Ok_Cancel2 :dialog
{
  label = " 面積計算 ";
  key = "menseki";
  spacer;
  :row     /* タイルを横並びに配置スタート * /
  {
    :edit_box   /* 編集ボックス処理タイルスタート */
    {
      label = " 縦 ";
      key = "tate";
      width = 6;
      value = "";
    }        /* 編集ボックス処理タイル終了 */
    :edit_box   /* 編集ボックス処理タイルスタート */
    {
      label = " 横 ";
      key = "yoko";
      width = 6;
      value = "";
    }        /* 編集ボックス処理タイル終了 */
  }          /* タイルを横並びに配置終了 */
  spacer;
  /* OK or CANCEL */
  :row     /* タイルを横並びに配置スタート * /
  {
    : button
    {
       label = "Ok";
       is_default = true;
       key = "accept";
       fixed_width = true;
    }
    : button
    {
       label = "Cancel";
       is_cancel = true;
       key = "Cancel";
       fixed_width = true;
    }
    : button
    {
       label = "Help";
       key = "help";
       fixed_width = true;
    }
  }    /* タイルを横並びに配置終了 */
  spacer;
}//end
```





使用する LISP	Ok_cancel2.lsp

前ページのダイアログの基本的な LISP 関数です。これを元に①から③のケースを説明します。

```
(defun c:Ok_Cancel2( / dialog-box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "Ok_Cancel2"))     ;Ok_Cancel2.dcl をメモリーに読み込む
  (new_dialog "Ok_Cancel2" dcl_id)             ;Ok_Cancel2 ダイアログを初期化する
                                                 → ダイアログの読み込みと表示

;<< tate の初期設定 >>
❷ (set_tile "tate" "")                         ;edit_box"tate" に ""(nil) をセット
  ;<< yoko の初期設定 >>
  (set_tile "yoko" "")                         ;edit_box"yoko" に ""(nil) をセット
                                                 → edit_box"tate" と "yoko" を初期化する
;<< edit_box 処理 tate に値を受け取る >>
❸ (action_tile "tate"                          ;edit_box"tate" にデータを入力
    (strcat
      "(setq tate (get_tile ¥"tate¥"))"        ;edit_box"tate" のデータを取得
      "(mode_tile ¥"yoko¥" 2)"                 ;edit_box"yoko" にフォーカスを移動
    )
  )                                              → edit_box"tate" のデータを取得する
;<< edit_box 処理 yoko に値を受け取る >>
❹ (action_tile "yoko"                          ;edit_box"yoko" にデータを入力
      "(setq yoko (get_tile ¥"yoko¥"))"        ;edit_box"yoko" のデータを取得
  )                                              → edit_box"yoko" のデータを取得する
;<< ダイアログを終了する準備 >>
❺ (setq dialog-box (start_dialog))             ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                       ; ダイアログをメモリーから解放する
                                                 → ダイアログの入力と解放
  (princ)
);end
```

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	edit_box "tate" と "yoko" に初期値を入力します。[set_tile] は [load_dialog] の後、[action_tile] の前に記述します。ここでは、""（空白）を指定していますが、既定値を指定することも出来ます。
③	edit_box "tate" の内容を変数 "tate" に代入します。（文字列） modo_tile で次のタイル "yoko" にカーソルが移動します。(Tab キーと同じ働きです。)
④	edit_box "yoko" の内容を変数 "yoko" に代入します。（文字列）
⑤	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値 <status> を返します。ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。

タイルをコントロール

① サブ関数（コールバック関数）を使わずに、AutoCAD のメッセージボックスに表示する。

第 1 手法 システム変数 [DIASTAT] の値に従って、メッセージボックスに表示します。

```
;<< 面積の数値を変数 [goukei] にセットする >>
 (setq goukei (* (atof tate) (atof yoko)))
 (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
;<< ダイアログを終了する >>
 (setq dialog-box (start_dialog))
 (unload_dialog dcl_id)

(if(= (getvar "DIASTAT") 1)
  (alert goukei_moji)
  )
```

→ Step1
→ Step2
→ Step3

Step1・・(setq goukei (* (atof tate) (atof yoko)))
→ edit_box "tate" と edit_box "yoko" の情報は文字ですから、計算処理のために実数に変換し、乗算して goukei にセットします。

・・(setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
→ alert 関数で表示する情報は文字列ですから、実数の goukei を文字に変換 (rtos) して、" 面積 :" と (rtos goukei) と " です " を strcat 関数で 1 つの文字列に結合します。

Step2・・(setq dialog-box (start_dialog))
→ダイアログにユーザーの入力を受け付けます。

・・(unload_dialog dcl_id)
→ダイアログをメモリーから解放します。

Step3・・(if(= (getvar "DIASTAT") 1)
→ダイアログを [OK(accept)] で終了したときは、システム変数 [DIASTAT] に <1> がセットされています。

・・(alert goukei_moji)
→もし、[OK] ボタンが押されたときは、alert 関数で指定した文字列を表示します。

第 2 手法 [done_dialog] の値 <status> に従って、メッセージボックスに表示します。

```
;<< 面積の数値を変数 [goukei] にセットする >>
 (setq goukei (* (atof tate) (atof yoko)))
 (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
;<< ダイアログを終了する >>
 (action_tile "accept" "(done_dialog 1)")
 (action_tile "cancel" "(done_dialog 0)")
 (setq chkdia (start_dialog))
 (unload_dialog dcl_id)

 (if (= chkdia 1)
   (alert goukei_moji)
 );if
```

→ Step1
→ Step2
→ Step3

Step1・・(action_tile "accept" "(done_dialog 1)")
→ [OK(accept)] で終了したときは、フラッグ (status)<1> をたてて、ダイアログを終了します。

・・(action_tile "cancel" "(done_dialog 0)")
→ [Cancel(cancel)] で終了したときは、フラッグ (status)<0> をたてて、ダイアログを終了します。

Step2・・(setq chkdia (start_dialog))
→ 終了したときの返り値 [status] の情報を、変数 [chkdia] にセットします。

・・(unload_dialog dcl_id)
→ ダイアログをメモリーから解放します。

Step3・・(if (= chkdia 1)
→返り値 [chkdia] が <1> か <0> かを判定します。

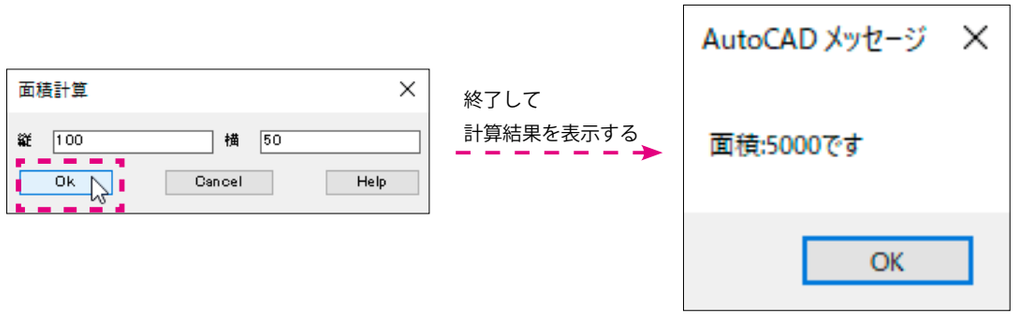
・・(alert goukei_moji)
→ もし、<1> であれば、alert 関数で指定した文字列を表示します。

Point!

第 1 手法と第 2 手法は同じように見えますが、
第 1 手法では、ダイアログが終了した後に、[Ok] で終了したかどうかを判定しています。このケースでは、再度ダイアログに戻って計算処理を行うことはできません。
第 2 手法では、(done_dialog 1) の前に次ページの 2 のように (GetTileVal) などの内部サブ関数を追加で記述することが出来ます。

② [Ok] ボタンから内部サブ関数を使用して、AutoCAD のメッセージボックスに表示する。

第 1 手法 [OK] ボタンを押したときに、メッセージを表示してダイアログを閉じる。



内部サブ関数

```
;<< タイルの数値をゲットする（内部サブ関数）>>
 (defun GetTileVal ()
;<< 面積の数値を変数 [goukei] にセットする >>
  (setq goukei (* (atof tate) (atof yoko)))
  (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
  (alert goukei_moji)
 )
;<< [Ok] ボタンが選択された時の処理を記述する >>
(action_tile "accept" "(GetTileVal)(done_dialog 1)")
(action_tile "cancel" "(done_dialog 0)")
;<< ダイアログを終了する >>
(setq chkdia (start_dialog))
(unload_dialog dcl_id)
```

→ Step2

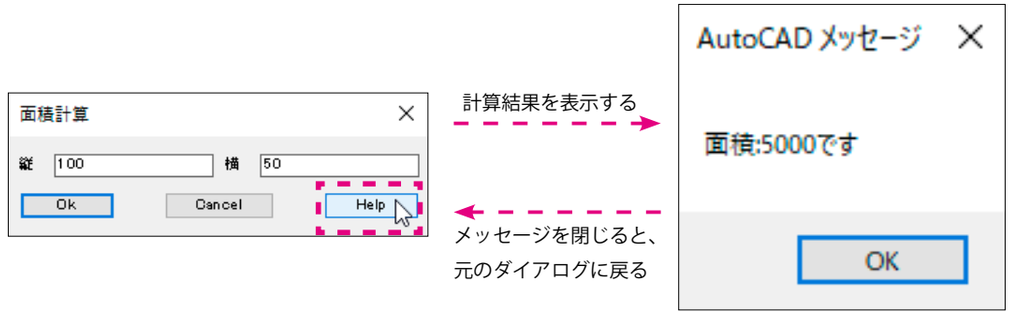
→ Step1

Step1・・(action_tile "accept" "(GetTileVal)(done_dialog 1)")
→ [Ok] ボタンを押してダイアログを閉じる前に、内部サブ関数 (GetTileVal) を実行します。

Step2・・(defun GetTileVal ()・・
→計算を処理する関数 (GetTileVal) を作成して、start_dialog の前に記述します。
処理する行数が少ない場合は、1 のようにサブ関数を使わなくて済みますが、行数が多い場合は、独立させた方がメンテナンスが容易です。

② [help] ボタンから内部サブ関数を使用して、AutoCAD のメッセージボックスに表示する。

第 2 手法 [Help] ボタンを押したときに、メッセージを表示するがダイアログは閉じない。



内部サブ関数

```
;<< タイルの数値をゲットする（内部サブ関数）>>
 (defun GetTileVal ()
;<< 面積の数値を変数 [goukei] にセットする >>
  (setq goukei (* (atof tate) (atof yoko)))
  (setq goukei_moji (strcat " 面積 :" (rtos goukei) " です "))
  (alert goukei_moji)
 )
;<< [help] ボタンが選択された時の処理を記述する >>
(action_tile "help" "(GetTileVal)")
;<< [Ok] ボタンと [Cancel] ボタンが選択された時の処理を記述する >>
(action_tile "accept" "(done_dialog 1)")
(action_tile "cancel" "(done_dialog 0)")
;<< ダイアログを終了する >>
(setq chkdia (start_dialog))
(unload_dialog dcl_id)
```

前ページと同じ
→ Step2

→ Step1

Step1・・(action_tile "help" "(GetTileVal)")
→内部サブ関数 (GetTileVal) を起動するボタンを [Help] キーに割り当てています。
この場合は、[Ok] ボタン <"accept"> キーは押されていないので、ダイアログが閉じることはありません。

Point!

ユーザーのヘルプを help ボタンに割り当てる場合は、
一例として (action_tile "help" "(help01)") と記述します。
<help**> と番号を付けて、必要に応じて呼び出します。

☞P2-98 の⑪を参考

③ 外部サブ関数を使用して、AutoCAD のメッセージボックスに表示する。

第 1 手法 システム変数 <DIASTAT> の値で、外部サブ関数を分岐する。

外部サブ関数

```
;<< ダイアログを終了する >>
 (setq dialog-box (start_dialog))
 (unload_dialog dcl_id)

 (if(= (getvar "DIASTAT") 1) (Ok_Cancel2_1))   → Step1
 (princ)
) ;dialog_end
;<< 外部サブ関数を起動 >>
(defun Ok_Cancel2_1 ()                          → Step2
 (alert goukei_moji)
)
```

Step1・・(if(= (getvar "DIASTAT") 1) (Ok_Cancel2_1))
　　→ダイアログを [OK(accept)] で終了したときは、外部サブ関数 (Ok_Cancel2_1) を起動。
Step2・・alert 関数で指定した文字列を表示します。

第 2 手法 "done_dialog" が返す値で、外部サブ関数を分岐する。

外部サブ関数

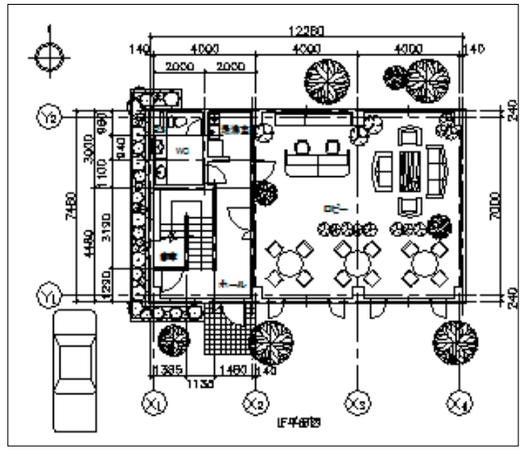
```
;<< [Ok] ボタンと [Cancel] ボタンが選択された時の処理を記述する >>
(action_tile "accept" "(done_dialog 1)")
(action_tile "cancel" "(done_dialog 0)")
 ;<< ダイアログを終了する >>
(setq chkdia (start_dialog))
(unload_dialog dcl_id)

(if (= chkdia 1)(Ok_Cancel2_1))   → Step1
(princ)
) ;dialog_end
;<< 外部サブ関数を起動 >>
(defun Ok_Cancel2_1 ()            → Step2
 (alert goukei_moji)
)
```

Step1・・(if (= chkdia 1)(Ok_Cancel2_1)
　　→フラッグ (status)[chkdia] が <1> であれば、外部サブ関数 (Ok_Cancel2_1) を起動。
Step2・・alert 関数で指定した文字列を表示します。

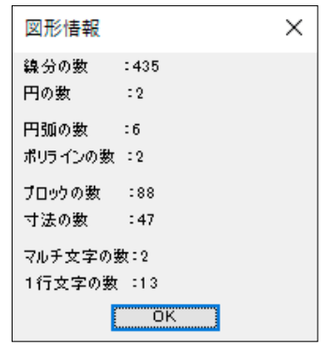
第 2 節　図形の数を表示する

使用する LISP	S_Text2.lsp	使用する主な関数	ssget、sslength、strcat、itoa
内部サブ関数	なし	外部サブ関数	(S_Text2_1)
使用する DCL	S_Text2.dcl	使用するタイル	text_tile
外部テキスト	なし	スライドファイル	なし
このプログラムは、図面にある図形の数を表示します。 図面範囲やオブジェクト範囲などを追加すれば、AutoCAD の [図面情報 <status>] コマンドと同様の機能に拡張できます。			

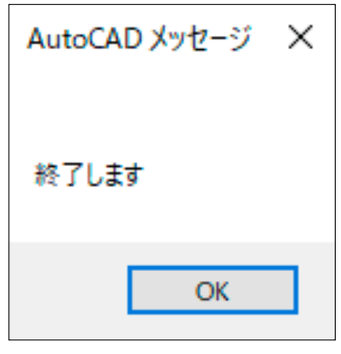
Step1 ー <S_Text2.lsp> を起動すると、右下図のダイアログが表示されます。
　　ここでは図形の数だけですが、画層名やブロック名の一覧などを追加することも可能です。



プログラム実行後の表示

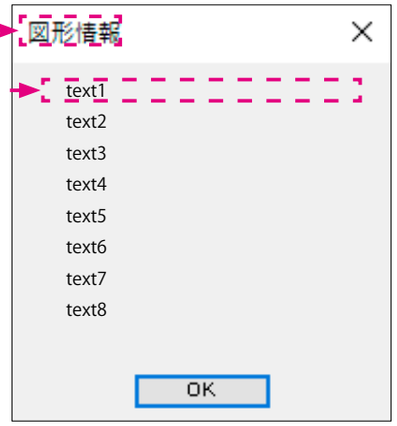


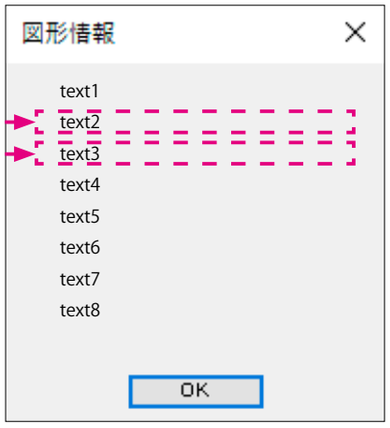
Step2 ー上記ダイアログで [OK] ボタンを押すと、< 終了します > のメッセージが表示されます。
　　[OK] ボタンを押して、このメッセージ ボックスを閉じます。

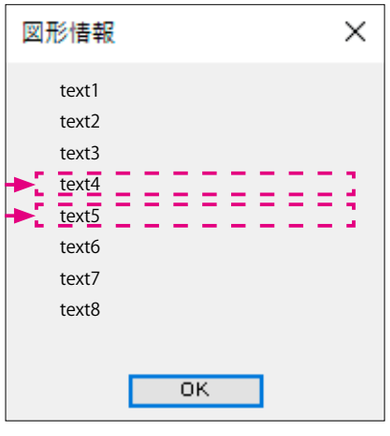


使用するダイアログ	S_Test2.dcl

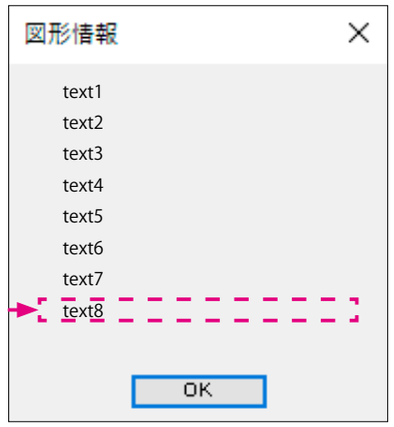
```
S_Text2 : dialog
{
 label = " 図形情報 ";
 : column    /* タイルを縦並びに配置スタート */
 {
  : text    /* 線分の数 */
  {
   label = "";
   key = "text1";
   width = 30;
  }
  : text    /* 円の数 */
  {
   label = "";
   key = "text2";
   width = 30;
  }
 spacer;
 : text    /* 円弧の数 */
 {
  label = "";
  key = "text3";
  width = 30;
 }
 : text    /* ポリラインの数 */
 {
  label = "";
  key = "text4";
  width = 30;
 }
spacer;
 : text    /* ブロックの数 */
 {
  label = "";
  key = "text5";
  width = 30;
 }
```



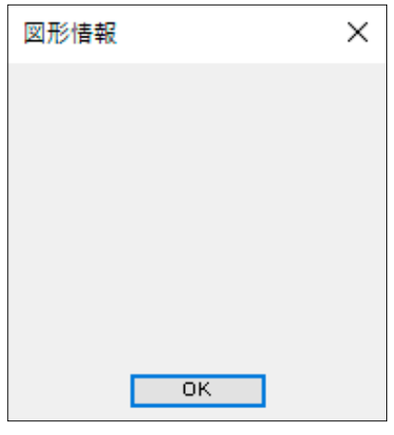




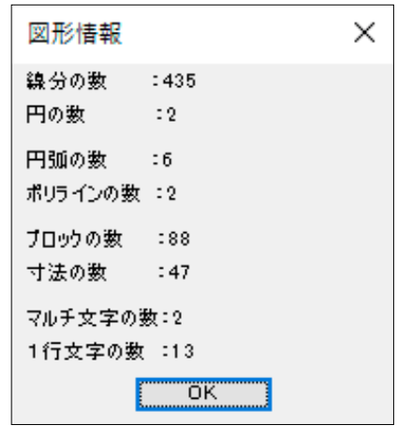

```
 : text    /* 寸法の数 */
 {
  label = "";
  key = "text6";
  width = 30;
 }
 spacer;
 : text    /* マルチ文字の数 */
 {
  label = "";
  key = "text7";
  width = 30;
 }
 : text    /*1 行文字の数 */
 {
  label = "";
  key = "text8";
  width = 30;
 }
}    /* タイルを縦並びに配置終了 */
 ok_only;

}//end
```



作成時の状態



起動後の状態



タイルをコントロール

使用する LISP	S_Test2.lsp

```
(defun c:S_Text2 ( / lin1 cir1 arc1 pol1 ins1 dim1 mte1 tex1)
; << 線分の個数を数える >>
 (setq lin1 (ssget "x" '((0 . "LINE"))))  ;ssget 関数で [LINE] だけを変数 <lin1> にセット
 (setq num_lin (sslength lin1))  ;sslength 関数で、変数 <lin1> の数を数える
;<< 円の個数を数える >>
 (setq cir1 (ssget "x" '((0 . "CIRCLE"))))  ;ssget 関数で [CIRCLE] だけを変数 <cir1> にセット
 (setq num_cir (sslength cir1))  ;sslength 関数で、変数 <cir1> の数を数える
;<< 円弧の個数を数える >>
 (setq arc1 (ssget "x" '((0 . "ARC"))))  ;ssget 関数で [ARC] だけを変数 <arc1> にセット
 (setq num_arc (sslength arc1))  ;sslength 関数で、変数 <arc1> の数を数える
; << ポリラインの個数を数える >>
 (setq pol1 (ssget "x" '((0 . "LWPOLYLINE"))))  ;ssget 関数で [LWPOLYLINE] だけを変数 <pol1> にセット
 (setq num_pol (sslength pol1))  ;sslength 関数で、変数 <pol1> の数を数える
; << ブロックの個数を数える >>
 (setq ins1 (ssget "x" '((0 . "INSERT"))))  ;ssget 関数で [INSERT] だけを変数 <ins1> にセット
 (setq num_ins (sslength ins1))  ;sslength 関数で、変数 <ins1> の数を数える
; << 寸法の個数を数える >>
 (setq dim1 (ssget "x" '((0 . "DIMENSION"))))  ;ssget 関数で [DIMENSION] だけを変数 <dim1> にセット
 (setq num_dim (sslength dim1))  ;sslength 関数で、変数 <dim1> の数を数える
; << マルチテキストの個数を数える >>
 (setq mte1 (ssget "x" '((0 . "MTEXT"))))  ;ssget 関数で [MTEXT] だけを変数 <mte1> にセット
 (setq num_mte (sslength mte1))  ;sslength 関数で、変数 <mte1> の数を数える
; <<1 行文字の個数を数える >>
 (setq tex1 (ssget "x" '((0 . "TEXT"))))  ;ssget 関数で [TEXT] だけを変数 <tex1> にセット
 (setq num_tex (sslength tex1))  ;sslength 関数で、変数 <tex1> の数を数える
```

☞ssget 関数の使い方は、P1-247 ～ P1-251 を参照

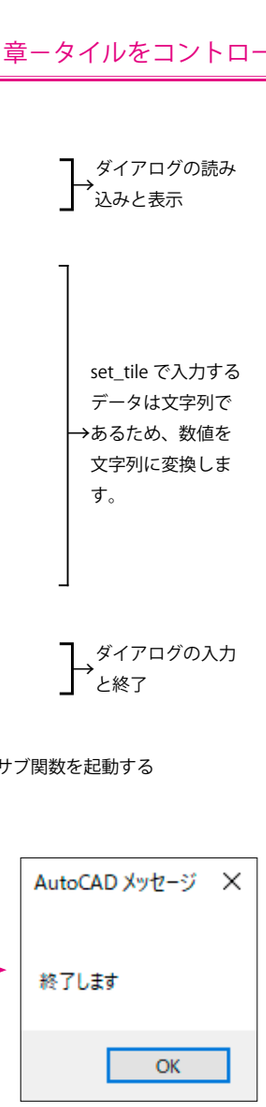
Point!

ssget 関数で図形の取得方法

- 図面内のすべての図形を取得する
  → (ssget "X")
- 特定の図形 (Circle) だけを取得する
  → (ssget "X" '((0 . "CIRCLE")))
- 特定の画層 <0> の特定の図形 (Circle) だけを取得する
  → (ssget "X" '((0 . "CIRCLE") (8 . "0")))
- 半径が <10.0> 以下の円だけを取得する
  → (ssget "X" '((0 . "CIRCLE") (-4 . "<=") (40 . 10.0)))


```
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Text2"))  ;S_Text2.dcl をメモリーに読み込む
  (new_dialog "S_Text2" dcl_id)          ;S_Text2 ダイアログを初期化する
;<< 各タイルに図形の数を表示する >>
❷ (set_tile "text1" (strcat " 線分の数          : " (itoa num_lin)))
  (set_tile "text2" (strcat " 円の数            : " (itoa num_cir)))
  (set_tile "text3" (strcat " 円弧の数          : " (itoa num_arc)))
  (set_tile "text4" (strcat " ポリラインの数    : " (itoa num_pol)))
  (set_tile "text5" (strcat " ブロックの数      : " (itoa num_ins)))
  (set_tile "text6" (strcat " 寸法の数          : " (itoa num_dim)))
  (set_tile "text7" (strcat " マルチ文字の数    : " (itoa num_mte)))
  (set_tile "text8" (strcat "  1 行文字の数     : " (itoa num_tex)))
;<< ダイアログを終了する準備 >>
❸ (setq dialog-box (start_dialog))  ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)            ; ダイアログをメモリーから解放する
;<<[OK] ボタンが押されたら、外部サブ関数 (S_text2_1) を起動する >>
❹ (if(= (getvar "DIASTAT") 1) (S_Text2_1))  ;[OK] ボタンが押された時、外部サブ関数を起動する
(princ)
);end
;<< これ以下は、外部サブ関数 >>
❺ (defun S_Text2_1()
   (alert "¥n 終了します ")
  );end
```

;[OK] ボタンが押された時に、この関数に移る

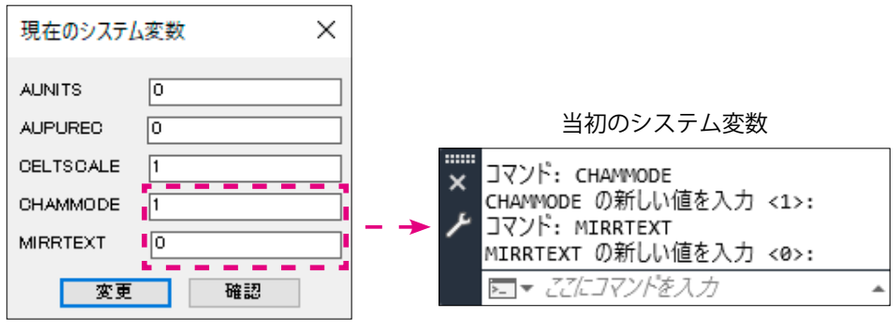


番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	"text1" ～ "text8" のタイル に初期値を入力します。[set_tile] は [load_dialog] の後、[action_tile] の前に記述します。ここでは、タイルに記述するデータは文字列なので、整数を文字列に変換する処理が必要です。
③	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値 (status) を返します。ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
④	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_Text2_1) を起動し、ダイアログを終了します。
⑤	外部サブ関数 (S_Text2_1) を起動します。この時点で、前のダイアログに戻ることはできません。

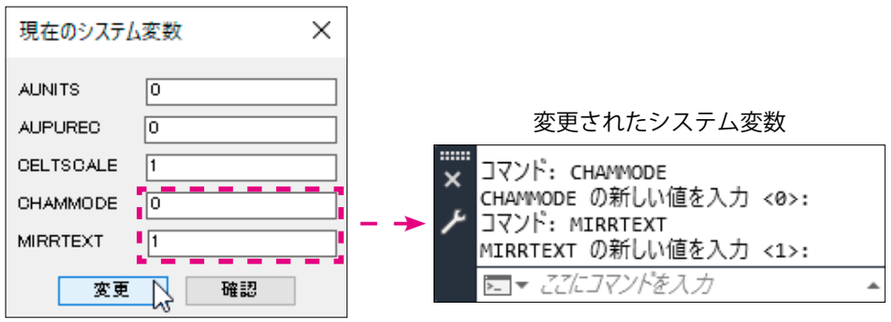
第 3 節　システム変数を読み込み、変更する

使用する LISP	S_Text1.lsp	使用する主な関数	getvar、itoa、rtos、atoi、atof
内部サブ関数	なし	外部サブ関数	(S_Text1_1)
使用する DCL	S_Text1.dcl	使用するタイル	edit_box
外部テキスト	なし	スライドファイル	なし
このプログラムは、システム変数を一覧表示した後、変更した変数の書き込み（再設定）を行います。 このようなプログラムを使用すると、システム変数の変更をまとめて処理できます。			

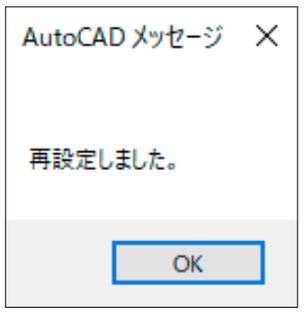
Step1 ー <S_Text1.lsp> を起動すると、下図のダイアログが表示されます。
この時点では、現在のシステム変数の状態が 5 つ表示されます。



Step2 ー変更するシステム変数のボックスに新しい数値を入力し [変更] ボタンを押します。
[確認] ボタンを押すと、何も処理をしないで終了します。



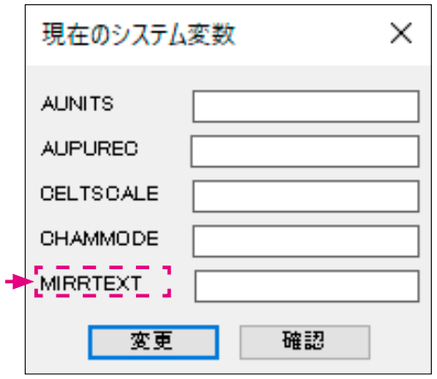
Step3 ー < 再設定しました > のメッセージが表示されます。[OK] ボタンを押して終了します。



使用するダイアログ	S_Test1.dcl

```
S_Text1 : dialog
{
  label = " 現在のシステム変数 " ;
  spacer;
  : column    /* タイルを縦並びに配置スタート */
  {
       :edit_box   /* 編集ボックス処理タイルスタート */
        {
          label = "AUNITS     ";
          key = "aun";
          width = 3;
          value = "";
        }        /* 編集ボックス処理タイル終了 */
       :edit_box   /* 編集ボックス処理タイルスタート */
        {
          label = "AUPUREC    ";
          key = "aup";
          width = 3;
          value = "";
        }        /* 編集ボックス処理タイル終了 */
      :edit_box   /* 編集ボックス処理タイルスタート */
       {
          label = "CELTSCALE  ";
          key = "cel";
          width = 3;
          value = "";
        }        /* 編集ボックス処理タイル終了 */
      :edit_box   /* 編集ボックス処理タイルスタート */
       {
          label = "CHAMMODE";
          key = "cha";
          width = 3;
          value = "";
        }        /* 編集ボックス処理タイル終了 */
```

```
        :edit_box  /* 編集ボックス処理タイルスタート */
        {
          label = "MIRRTEXT    ";
          key = "mir";
          width = 3;
          value = "";
        }        /* 編集ボックス処理タイル終了 */
}    /* タイルを縦並びに配置終了 */
spacer;
: row    /* タイルを横並びに配置スタート */
{
 fixed_width = true;
 alignment = centered;
 : ok_button
 {
  label = " 変更 ";
  width = 11;
 }
 : cancel_button
 {
  label = " 確認 ";
  width = 11;
 }
}    /* タイルを横並びに配置終了 */
}//end
```



Point!

[Ok_Cancel;] では、ラベルを付けることはできません。
ボタンにラベルを付けるためには、①か②の方法になります。

① :ok_button
```
  {
   label = " 変更 ";
   width = 11
  }
```

② :button
```
  {
   label = " 変更 ";
   is_default = true;
   key = "accept";          ←必須です。
   width = 8;
   fixed_width = true;
  }
```

①のタイプは、AutoCAD によって、[Ok<accept>] ボタンであることが指定されていますが、
②のようにユーザーが独自で button を作成したときは、[key = "accept";] の記述が必要です。

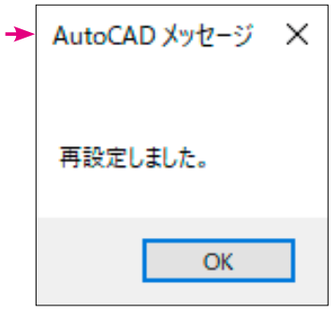
使用する LISP	S_Text1.lsp

```
(defun c:S_Text1 ( / aun_mode aup_mode cel_mode cha_mode  mir_mode dcl_id
                     dialog-boxn_aun_mode  n_aup_mode n_cel_mode n_cha_mode
                     n_mir_mode)
; << システム変数を取得 >>
❶ (setq aun_mode (getvar "AUNITS"))          ; システム変数 <AUNITS> を取得する
  (setq aup_mode (getvar "AUPREC"))          ; システム変数 <AUPREC> を取得する
  (setq cel_mode (getvar "CELTSCALE"))       ; システム変数 <CELTSCALE> を取得する
  (setq cha_mode (getvar "CHAMMODE"))        ; システム変数 <CHAMMODE> を取得する
  (setq mir_mode (getvar "MIRRTEXT"))        ; システム変数 <MIRRTEXT> を取得する
                                              → 最初に現在のシステム変数を取得しておく
;<< 値を変数にセットする >>
❷ (setq aun_mode (itoa aun_mode))            ; 整数を文字列に変換
  (setq aup_mode (itoa aup_mode))            ; 整数を文字列に変換
  (setq cel_mode (rtos cel_mode))            ; 実数を文字列に変換
  (setq cha_mode (itoa cha_mode))            ; 整数を文字列に変換
  (setq mir_mode (itoa mir_mode))            ; 整数を文字列に変換
                                              → タイルに表示するデータは文字なので、数値を文字に変換する
;<< ダイアログを使用する準備 >>
❸ (setq dcl_id (load_dialog "S_Text1"))      ;S_Text1.dcl をメモリーに読み込む
  (new_dialog "S_Text1" dcl_id)              ;S_Text1 ダイアログを初期化する
                                              → ダイアログの読み込みと表示
;<< ダイアログの edit_box に値をセットする >>
❹ (set_tile "aun" aun_mode)                  ;edit_box"aun" に変数 <aun_mode> を代入
  (set_tile "aup" aup_mode)                  ;edit_box"aup" に変数 <aup_mode> を代入
  (set_tile "cel" cel_mode)                  ;edit_box"cel" に変数 <cel_mode> を代入
  (set_tile "cha" cha_mode)                  ;edit_box"cha" に変数 <cha_mode> を代入
  (set_tile "mir" mir_mode)                  ;edit_box"mir" に変数 <mir_mode> を代入
                                              → 文字に変換したシステム変数をタイルにセットする
;<<[ 変更 ] ボタンが押された時に、edit_box の値を取得する >>
❺ (defun SaveTileVal()   ; コールバック関数
  (setq n_aun_mode (get_tile "aun"))         ;edit_box"aun" の値を取得
  (setq n_aup_mode (get_tile "aup"))         ;edit_box"aup" の値を取得
  (setq n_cel_mode (get_tile "cel"))         ;edit_box"cel" の値を取得
  (setq n_cha_mode (get_tile "cha"))         ;edit_box"cha" の値を取得
  (setq n_mir_mode (get_tile "mir"))         ;edit_box"mir" の値を取得
)                                             → edit_box の最終データを取得する
```

Point!

①と②は③の前に処理しておきます。
③と⑦の間に④⑤⑥を記述します。
④の (set_tile) の後に⑤の (get_tile) と⑥の (action_tile) を記述します。

```
❻ (action_tile "accept" "(SaveTileVal)(done_dialog 1)") ;[ 変更 ] ボタンが押された時の処理
  (action_tile "cancel" "(done_dialog 0)")              ;[ 確認 ] ボタンが押されると終了
;<< ダイアログを終了する準備 >>
❼ (setq chkdia (start_dialog))  ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)        ; ダイアログをメモリーから解放する
                                                        ]→ ダイアログの入力と終了
;<< プログラムの分岐 >>
❽ (if(= chkdia 1)               ;[ 変更 ] ボタンが押された (= chkdia 1) 時は、外部サブ関数 (S_Text1_1) を起動する
    (S_Text1_1)
  )
  (princ)
  );end

❾ (defun S_Text1_1()  ; システム変数を setvar 関数で再設定する
    (setvar "AUNITS" (atoi n_aun_mode))          ;n_aun_mode の値を整数に変換する
    (setvar "AUPREC" (atoi n_aup_mode))          ;n_aup_mode の値を整数に変換する
    (setvar "CELTSCALE" (atof n_cel_mode))       ;n_cel_mode の値を実数に変換する
    (setvar "CHAMMODE" (atoi n_cha_mode))        ;n_cha_mode の値を整数に変換する
    (setvar "MIRRTEXT" (atoi n_mir_mode))        ;n_mir_mode の値を整数に変換する
    (alert "¥n 再設定しました。")
    (princ)
    );end
```

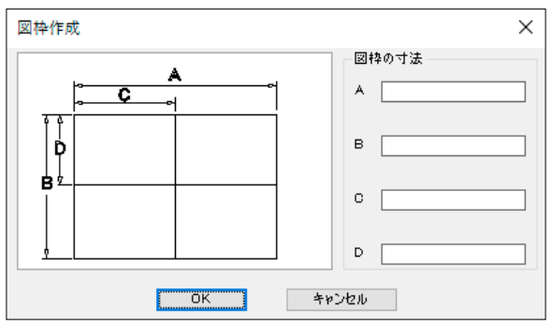


番号	説　明
①	最初にシステム変数の値を取得しておきます。その値を set_tile で各タイルにセットします。
②	タイルに記述するデータは文字列なので、整数や実数を文字列に変換する処理が必要です。
③	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
④	文字列に変換したシステム変数の値を各タイルに初期値としてセットします。
⑤	[変更] ボタンが押されると、この内部サブ関数が起動します。 各タイルの最後の値を取得します。
⑥	[変更] ボタンが押されると、内部サブ関数 (SaveTileVal) を起動して、フラッグ (status) として <1> を指定します。この数値はユーザーが指定できます。
⑦	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑧	フラッグ (status) が <1> であれば、外部サブ関数 (S_Text1_1) を起動します。<0> であれば何もせずに終了します。
⑨	各タイルの情報は文字列なので、整数や実数に変換してシステム変数を再設定しています。

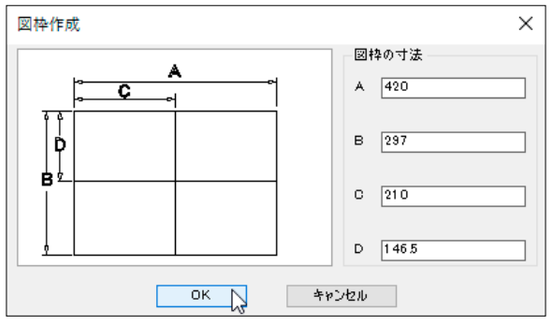
第 4 節　図枠の数値を入力する

使用する LISP	S_Edit1.lsp	使用する主な関数	getvar、atof、polar、ssadd、entlast
内部サブ関数	なし	外部サブ関数	(S_Edit1_1)
使用する DCL	S_Edit1.dcl	使用するタイル	icon_image、edit_box
外部テキスト	なし	スライドファイル	S_Edit1
このプログラムは、ダイアログの edit_box に直接数値を入力して、図枠を作図します。 図枠の大きさごとに作成しておくと、作業効率が上がります。			

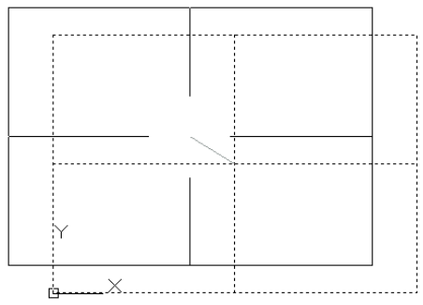
Step1 ー <S_Edit1.lsp> を起動すると、下図のダイアログが表示されます。
寸法の位置を画像で確認できるので、間違いも無くなります。



Step2 ー <A> から <D> の各ボックスに数値を入力して、[OK] ボタンを押します。

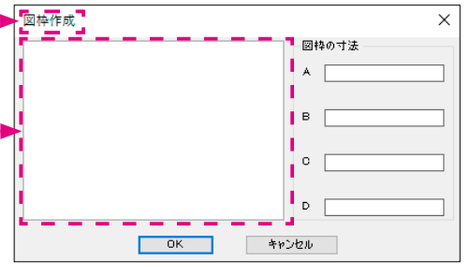


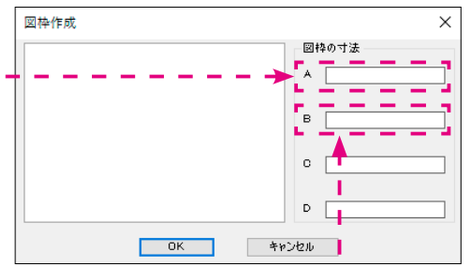
Step3 ー < 左図 > 図枠の挿入位置は、マウスで自由に変更できます。
< 右図 > 挿入した結果です。



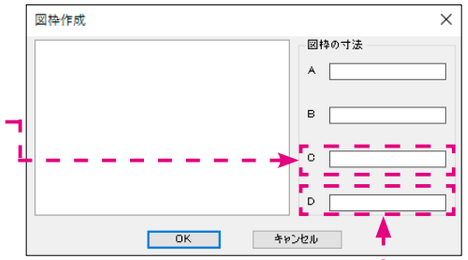
使用するダイアログ	S_Edit1.dcl

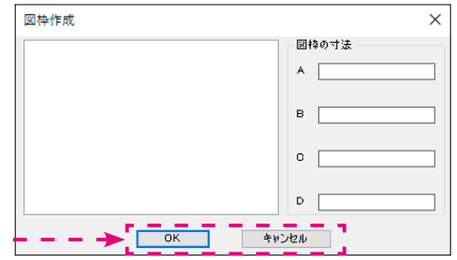
```
S_Edit1 :dialog
{
  label = " 図枠作成 ";
  :row              /* タイルを横並びに配置スタート */
  {
     :icon_image    /* イメージ処理タイルスタート */
     {
       label = "";
       key = "image00";
       width = 40;
       height = 15;
     }              /* イメージ処理タイル終了 */

    :boxed_column      /* タイルを縦並びに配置スタート */
    {
       label = " 図枠の寸法 ";

       :edit_box  /* 編集ボックス処理タイルスタート */
       {
         label = "A";
         key = "yoko1";
         width = 6;
         value = "";
       }        /* 編集ボックス処理タイル終了 */

       :edit_box  /* 編集ボックス処理タイルスタート */
       {
         label = "B";
         key = "tate1";
         width = 6;
         value = "";
       }        /* 編集ボックス処理タイル終了 */
```






```
       :edit_box  /* 編集ボックス処理タイルスタート */
       {
         label = "C";
         key = "yoko2";
         width = 6;
         value = "";
       }        /* 編集ボックス処理タイル終了 */

       :edit_box  /* 編集ボックス処理タイルスタート */
       {
         label = "D";
         key = "tate2";
         width = 6;
         value = "";
       }        /* 編集ボックス処理タイル終了 */
    }           /* タイルを縦並びに配置終了 */
  }             /* タイルを横並びに配置終了 */

/* OK or CANCEL */
    spacer;
    ok_cancel;
}//end
```



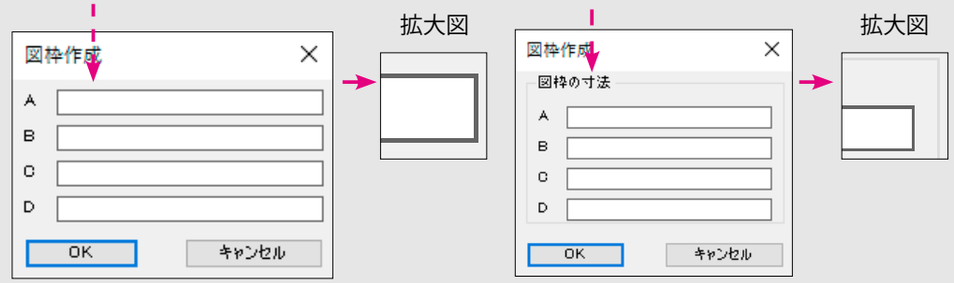


Point!

column や row にラベルを付けない場合（左）とラベルを付けた場合（右）

```
:column
{
 label = "";
```

```
:boxed_column
{
 label = " 図枠の寸法 ";
```



A から D 全体に外枠は作成できません。

A から D 全体に外枠が作成されます。

使用する LISP	S_Edit1.lsp

```
(defun c:S_Edit1( / dialog-box dcl_id x1 y1 yoko1 tate1 yoko2 tate2)
```

❶
```
(setq dcl_id (load_dialog "S_Edit1"))   ;S_Edit1.dcl をメモリーに読み込む
(new_dialog "S_Edit1" dcl_id)           ;S_Edit1 ダイアログを初期化する
```
→ ダイアログの読み込みと表示

```
;<< スライドを表示する準備 >>
```

❷
```
(setq x1 0 y1 0)                                ; スライドを表示する位置（原点は左上）
(setq x2 (dimx_tile "image00"))                 ; スライドを表示する X の幅（右方向）
(setq y2 (dimy_tile "image00"))                 ; スライドを表示する Y の幅（下方向）
(start_image "image00")                         ;"image00" にスライドを読み込む
(slide_image x1 y1 x2 y2 "S_Edit1");            ; スライド (S_Edit1) を表示
(end_image)                                     ; スライドの読み込みを終了
```
→ スライドを表示する X と Y の幅として "image00" の幅を指定しています。

```
;<< 各タイル の初期設定 >>
```

❸
```
(set_tile "tate1" "")   ;"tate1" の初期値を空白 <""> にセット
(set_tile "tate2" "")   ;"tate2" の初期値を空白 <""> にセット
(set_tile "yoko1" "")   ;"yoko1" の初期値を空白 <""> にセット
(set_tile "yoko2" "")   ;"yoko2" の初期値を空白 <""> にセット
```
→ 4 つの edit_box の初期値をセット。ダイアログ側でセットしている場合は必要ありません。

```
;<< edit_box 処理 yoko1 に値を受け取る >>
```

❹
```
(action_tile "yoko1"
  (strcat
    "(setq yoko1 (get_tile ¥"yoko1¥"))"
    "(mode_tile ¥"tate1¥" 2)"
  )
)
```
;"yoko1" に値を入力する
; 以下の文字列を結合する
;edit_box "yoko1" の値を取得（文字）
;edit_box "tate1" にフォーカスを移動
→ edit_box "yoko1" が操作されたときの処理を記述します。

```
;<< edit_box 処理 tate1 に値を受け取る >>
```

❺
```
(action_tile "tate1"
  (strcat
    "(setq tate1 (get_tile ¥"tate1¥"))"
    "(mode_tile ¥"yoko2¥" 2)"
  )
)
```
;"tate1" に値を入力する
; 以下の文字列を結合する
;edit_box "tate1" の値を取得（文字）
;edit_box "yoko2" にフォーカスを移動
→ edit_box "tate1" が操作されたときの処理を記述します。

```
;<< edit_box 処理 yoko2 に値を受け取る >>
```

❻
```
(action_tile "yoko2"
  (strcat
    "(setq yoko2 (get_tile ¥"yoko2¥"))"
        "(mode_tile ¥"tate2¥" 2)"
  )
)
```
;"yoko2" に値を入力する
; 以下の文字列を結合する
;edit_box "yoko2" の値を取得（文字）
;edit_box "tate2" にフォーカスを移動
→ edit_box "yoko2" が操作されたときの処理を記述します。


```
;<< edit_box 処理 tate2 に値を受け取る >>
```

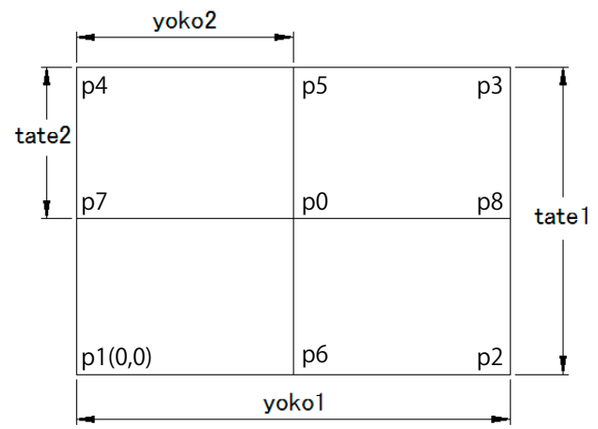
❼
```
(action_tile "tate2"
  (strcat
    "(setq tate2 (get_tile ¥"tate2¥"))"
  )
)
```
;"tate2" に値を入力する
; 文字列を結合する（1 行でも可）
;edit_box "tate2" の値を取得（文字）
→ edit_box "tate2" が操作されたときの処理を記述します。

❽
```
(setq dialog-box (start_dialog))   ; ユーザー入力を受け付ける
(unload_dialog dcl_id)             ; ダイアログをメモリーから解放する
```
→ ダイアログの入力と終了

```
;<<[OK] ボタンが押されたら、外部サブ関数 (S_Edit1_1) を起動する >>
```

❾
```
(if(= (getvar "DIASTAT") 1) (S_Edit1_1));[OK] ボタンが押された時のアクション
 (princ)
);end

;<< 外部サブ関数の定義 >>
(defun S_Edit1_1 ( / omode p0 p1 p2 p3 p4 p5 ent1 p6 p7 p8 ss1)
 (setq omode (getvar "OSMODE"))  ; システム変数 [OSMODE] の既定値を取得（最後に元に戻すため）
 (setvar "OSMODE" 0)  ; システム変数 [OSMODE] を <0> にセット
;<< 選択セットを用意 >>
 (setq ss1 (ssadd))  ; 作図した複数の図形を 1 度に選択するために、空の選択セットを事前に用意
;<<"edit_box" の値（文字列）を実数に変換 >>
(setq yoko1 (atof yoko1))  ;edit_box "yoko1" の値（文字列）を実数に変換
(setq tate1 (atof tate1))  ;edit_box "tate1" の値（文字列）を実数に変換
(setq yoko2 (atof yoko2))  ;edit_box "yoko2" の値（文字列）を実数に変換
(setq tate2 (atof tate2))  ;edit_box "tate2" の値（文字列）を実数に変換
```
→ edit_box の値は文字列なので、作図するために実数に変換します。

以下の順番で変数をセットしています。



```
;<< 外枠を描く >>
 (setq p1 '(0 0))  ; 外枠の左下を p1(0,0) にセット（最後にマウスで、p0 を基点に全体を移動します。）
 (setq p2 (polar p1 0 yoko1))  ;p2 は p1 から <0°（3 時方向）> で、距離は <yoko1> の位置
 (setq p3 (polar p2 (* 0.5 pi) tate1))  ;p3 は p2 から <90°（12 時方向）> で、距離は <tate1> の位置
 (setq p4 (polar p3 pi yoko1))  ;p4 は p3 から <180°（9 時方向）> で、距離は <yoko1> の位置
 (command "PLINE" p1 p2 p3 p4 "C")  ;[PLINE] コマンドで作図（"C" は p4 から p1 に結んで Close する意味）
 (setq ss1 (ssadd (entlast) ss1))  ; 作成した四角形 (entlast) を選択セット (ss1) に加える
;<< 内枠を描く >>
 (setq p5 (polar p4 0 yoko2))  ;p5 は p4 から <0°（3 時方向）> で、距離は <yoko2> の位置
 (setq p6 (polar p1 0 yoko2))  ;p6 は p1 から <0°（3 時方向）> で、距離は <yoko2> の位置
 (command "LINE" p5 p6 "")  ;[LINE] コマンドで作図
 (setq ss1 (ssadd (entlast) ss1))  ; 作成した線分 (entlast) を選択セット (ss1) に加える
 (setq p7 (polar p4 (* 1.5 pi) tate2))  ;p7 は p4 から <270°（6 時方向）> で、距離は <tate2> の位置
 (setq p8 (polar p3 (* 1.5 pi) tate2))  ;p8 は p3 から <270°（6 時方向）> で、距離は <tate2> の位置
 (command "LINE" p7 p8 "")  ;[LINE] コマンドで作図
 (setq ss1 (ssadd (entlast) ss1))  ; 作成した線分 (entlast) を選択セット (ss1) に加える
;<< 全体を移動する >>
 (setq p0 (polar p7 0 yoko2))  ;p7 から <0°（3 時方向）> で、距離は <yoko2> の位置を p0 にセット
 (command "MOVE" ss1 "" p0)  ;p0 を基点として、選択セット (ss1) を移動（目的点はユーザーの指示待ち）
;<<O スナップモードを元に戻す >>
 (setvar "OSMODE" omode)  ; システム変数 [OSMODE] を元の値に戻す
;<< グローバル変数の初期化 >>
 (setq tate1 nil tate2 nil yoko1 nil yoko2 nil)  ; グローバル変数をリセットする
);end
```

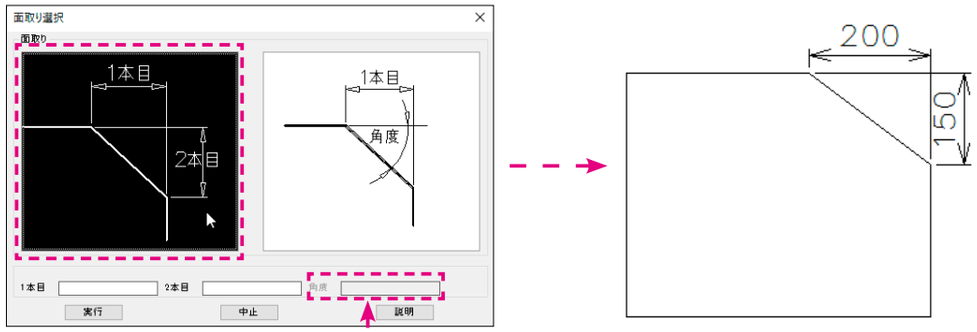
番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	スライドファイルを表示する範囲（X と Y の座標値）を指定します。縦横の幅を dcl で設定した "image00" の数値を指定しています。 イメージの表示は、[start_image][slide_image][end_image] の 3 つを記述します。
③	各 edit_box に初期値を入力します。[set_tile] は [load_dialog] の後、[action_tile] の前に記述します。
④	edit_box "yoko1" の内容を変数 "yoko1" に代入します。（文字列） modo_tile で次のタイル "tate1" にカーソルが移動します。(Tab キーと同じ働きです。）
⑤	edit_box "tate1" の内容を変数 "tate1" に代入します。（文字列） modo_tile で次のタイル "yoko2" にカーソルが移動します。(Tab キーと同じ働きです。）
⑥	edit_box "yoko2" の内容を変数 "yoko2" に代入します。（文字列） modo_tile で次のタイル "tate2" にカーソルが移動します。(Tab キーと同じ働きです。）
⑦	edit_box "tate2" の内容を変数 "tate2" に代入します。（文字列）
⑧	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑨	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_Edit1_1) を起動し、ダイアログを終了します。

第 5 節　画像でプログラムを選択する

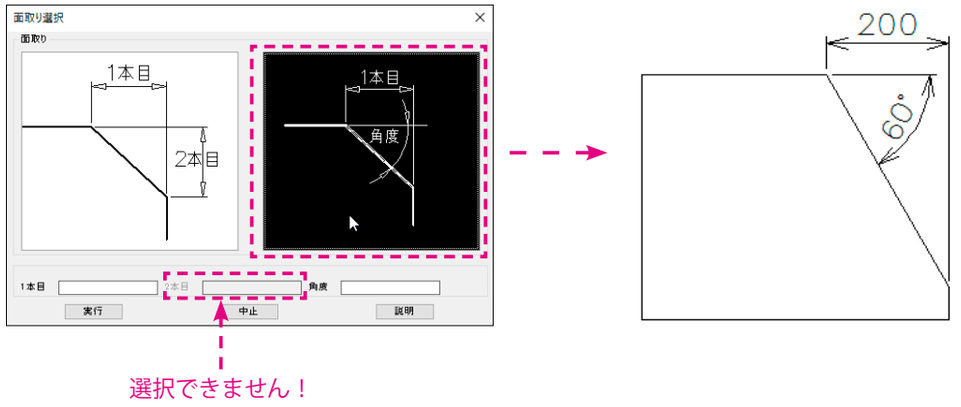
使用する LISP	S_Edit2.lsp	使用する主な関数	setvar、atof、angtof
内部サブ関数	(help00)	外部サブ関数	(S_Edit2_1)、(cham1)、(cham2)
使用する DCL	S_Edit2.dcl	使用するタイル	icon_image、edit_box
外部テキスト	なし	スライドファイル	S_Edit2-1、S_Edit2-2
このプログラムは、2 種類ある面取りのどちらかを選択して、面取りを行います。 面取りのタイプによって、システム変数 [CHAMMODE] の切り替えも自動で処理します。			

Step1 ー <S_Edit2.lsp> を起動すると、下図のダイアログが表示されます。
面取りのタイプを画像で選択できるので、視覚的に判りやすくなります。
また、入力不可のタイルを指定できますから、誤入力も無くなります。

左側の図を選択すると、[角度] のタイルが非表示になり、角度は入力できません。



Step2 ー右側の図を選択すると、[2 本目] のタイルが非表示になり、2 本目の入力はできません。

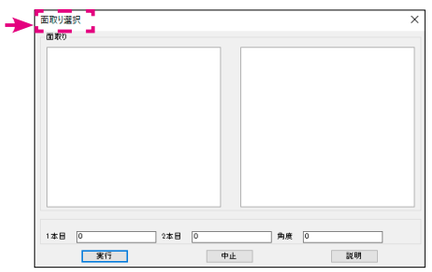


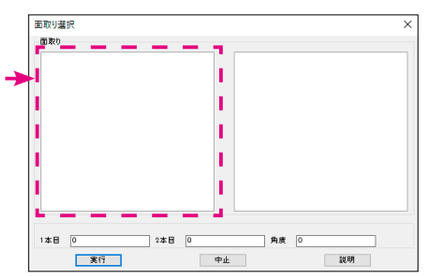
システム変数 [CHAMMODE] も自動的に切り替えています。

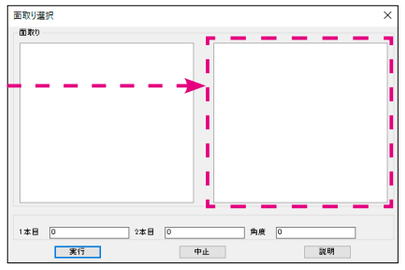
面取りモード	指定方法	システム変数
CHAMMODE<0>	2 つの面取り距離を指定します。	CHAMFERA、CHAMFERB
CHAMMODE<1>	面取り距離と角度を指定します。	CHAMFERC、CHAMFERD

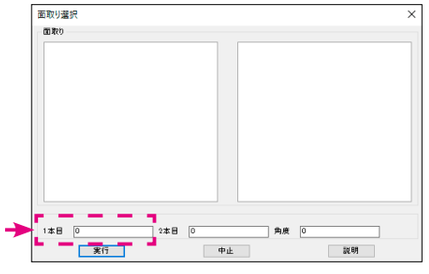
使用するダイアログ	S_Edit2.dcl

```
S_Edit2 : dialog
{
  label = " 面取り選択 ";

    :boxed_row  /* タイルを横並びに配置スタート */
    {
     label = " 面取り ";

      :icon_image /* イメージ処理タイルスタート */
      {
        label = "";
        key = "image01";
        width = 40;
        height =20;
      }        /* イメージ処理タイル終了 */

      :icon_image /* イメージ処理タイルスタート */
      {
        label = "";
        key = "image02";
        width = 40;
        height =20;
      }        /* イメージ処理タイル終了 */

    }          /* タイルを横並びに配置終了 */

    :boxed_row       /* タイルを横並びに配置スタート */
  {
      :edit_box  /* 編集ボックス処理タイルスタート */
      {
        label = "1 本目 ";
        key = "sen1";
        width = 6;
        fixed_width = true;
        value = 0;
      }        /* 編集ボックス処理タイル終了 */
```



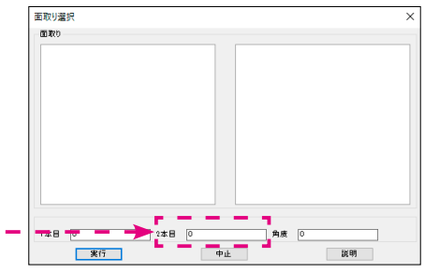


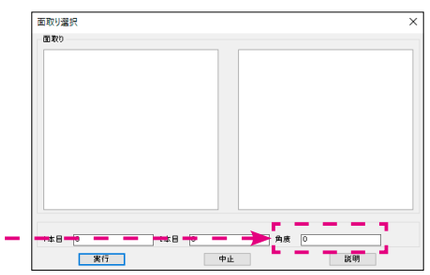




Point!


```
    :edit_box  /* 編集ボックス処理タイルスタート */
    {
      label = "2 本目 ";
      key = "sen2";
      width = 6;
      fixed_width = true;
      value = 0;
    }        /* 編集ボックス処理タイル終了 */

    :edit_box  /* 編集ボックス処理タイルスタート */
    {
      label = " 角度 ";
      key = "kaku";
      width = 6;
      fixed_width = true;
      value = 0;
    }        /* 編集ボックス処理タイル終了 */
  spacer;
  }    /* タイルを横並びに配置終了 */
```

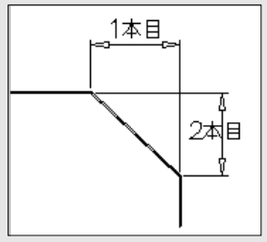




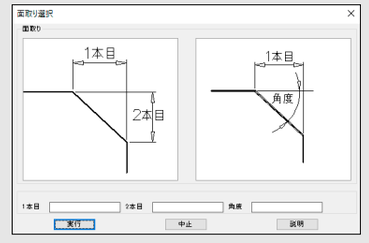
Point!

イメージタイルの大きさ (width と height) とスライドファイルの大きさの比率

使用するスライドファイル

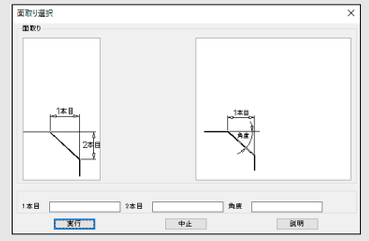


width = 40;
height =20;



width = 40;
height =20;

width = 20;
height =20;

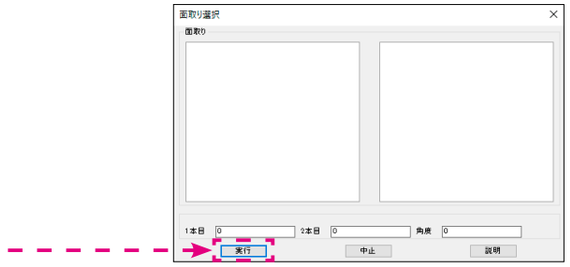


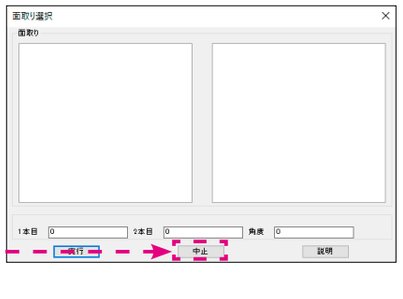
width = 40;
height =20;

右上のイメージタイルの大きさは、使用するスライドファイルの縦横の比率と同じにしていますが、右下のイメージタイルの大きさはスライドファイルの比率と違っています。

イメージタイルの縦横の数値を決めるときは、表示するスライドファイルの縦横の比率を考慮する必要があります。

```
/* OK or CANCEL */
 : row     /* タイルを横並びに配置スタート */
 {
   spacer;
   : button
   {
      label = " 実行 ";
      is_default = true;
      key = "accept";
      width = 8;
      fixed_width = true;
   }
    spacer;
    : button
   {
      label = " 中止 ";
      is_cancel = true;
      key = "Cancel";
      width = 4;
      fixed_width = true;
   }
    spacer;
    : button
   {
      label = "Help";
      key = "help";
      width = 4;
      fixed_width = true;
   }
   spacer;
  }    /* タイルを横並びに配置終了 */
}//end
```



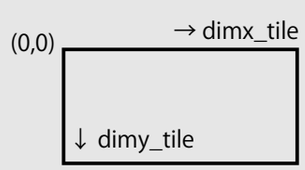


タイルをコントロール

使用する LISP	S_Edit2.lsp

```
(defun c:S_Edit2( / dialog_box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Edit2"))   ;S_Edit2.dcl をメモリーに読み込む
   (new_dialog "S_Edit2" dcl_id)          ;S_Edit2 ダイアログを初期化する
                                          → ダイアログの読み込みと表示

;<< イメージを表示する範囲を指定 >>   Point!
❷ (setq x1 0) (setq y1 0)                 ; スライドを表示する位置（原点は左上）
   (setq x2 (dimx_tile "image01")         ; スライドを表示する X の幅（右方向）
         y2 (dimy_tile "image01")         ; スライドを表示する Y の幅（下方向）
   )                                      → スライドを表示する X と Y の幅として "image01" の幅を指定しています。

;<< スライドを表示する準備 >>
❸ (start_image "image01")                 ;"image01" にスライドを読み込む
   (fill_image x1 y1 x2 y2 -2)            ; 背景色を現在の AutoCAD と同じにする
   (slide_image x1 y1 x2 y2 "S_Edit2-1")  ; スライド (S_Edit2-1) を表示
   (end_image)                            ; スライドの読み込みを終了
                                          → スライドを読み込み、背景色を指定して、"S_Edit2-1" のスライドを指定します。

;<< スライドを表示する準備 >>
❹ (start_image "image02")                 ;"image02" にスライドを読み込む
   (fill_image x1 y1 x2 y2 -2)            ; 背景色を現在の AutoCAD と同じにする
   (slide_image x1 y1 x2 y2 "S_Edit2-2")  ; スライド (S_Edit2-2) を表示
   (end_image)                            ; スライドの読み込みを終了
                                          → スライドを読み込み、背景色を指定して、"S_Edit2-2" のスライドを指定します。

;<< 各タイル の初期設定 >>
❺ (set_tile "sen1" "")                    ;edit_box "sen1" の初期値を空白 <""> にセット
   (set_tile "sen2" "")                   ;edit_box "sen2" の初期値を空白 <""> にセット
   (set_tile "kaku" "")                   ;edit_box "kaku" の初期値を空白 <""> にセット
                                          → 3 つの edit_box の初期値をセット。ダイアログ側でセットしている場合は必要ありません。

;<< edit_box 処理 sen1 に値を受け取る >>
❻ (action_tile "sen1"                     ;edit_box "sen1" を操作する
     "(setq sen1 (get_tile¥"sen1¥"))"     ;edit_box "sen1" の値を取得
   )                                      → edit_box "sen1" が操作されたときの処理を記述します。
```

Point! イメージを表示する範囲



イメージを表示する範囲の指定は、左上を原点 (0,0) として右下に数値が大きくなります。
横の幅を [dimx_tile] で指定し、縦の幅を [dimy_tile] で指定します。

(dimx_tile "image01")、(dimy_tile "image01") はダイアログで指定した "image01" の大きさを使用することを意味します。

タイルをコントロール

;<< edit_box 処理 sen2 に値を受け取る >>

```
(action_tile "sen2"                    ;edit_box "sen2" を操作
    "(setq sen2 (get_tile ¥"sen2¥"))"  ;edit_box "sen2" の値を取得
)
```

→ edit_box "sen2" が操作されたときの処理を記述します。

;<< edit_box 処理 kaku に値を受け取る >>

```
(action_tile "kaku"                    ;edit_box "kaku" を操作
    "(setq kaku (get_tile¥"kaku¥"))"   ;edit_box "kaku" の値を取得
)
```

→ edit_box "kaku" が操作されたときの処理を記述します。

;<< image01 処理 変数 check=1 >>

Point!

```
(action_tile "image01"                 ;icon_image "image01" を操作
  (strcat                              ; 以下の文字列を結合する
    "(setq check 1)"                   ; 変数 [check] の値を <1> にセット
    "(mode_tile ¥"kaku¥" 1)"           ;edit_box "kaku" を非アクティブ（入力不可能）にする
    "(mode_tile ¥"sen2¥" 0)"           ;edit_box "sen2" をアクティブ（入力可能）にする
  )
)
```

→ <1> で 2 本の線分の面取り指定

;<< image02 処理 変数 check=2 >>

Point!

```
(action_tile "image02"                 ;icon_image "image02" を操作
  (strcat                              ; 以下の文字列を結合する
    "(setq check 2)"                   ; 変数 [check] の値を <2> にセット
    "(mode_tile ¥"sen2¥" 1)"           ;edit_box "sen2" を非アクティブ（入力不可能）にする
    "(mode_tile ¥"kaku¥" 0)"           ;edit_box "kaku" をアクティブ（入力可能）にする
  )
)
```

→ <2> で線分と角度の面取り指定

```
(action_tile "help" "(help00)")  ; "help" ボタンが押された時、サブ関数 (help00) を起動
;<< ダイアログを終了する準備 >>
(setq dialog-box (start_dialog))  ; ユーザー入力を受け付ける
(unload_dialog dcl_id)            ; ダイアログをメモリーから解放する
```

→ ダイアログの入力と終了

```
;<<[ 実行 ] ボタンを押すと、システム変数 [DIASTAT] に <1> がセットされる >>
(if(= (getvar "DIASTAT") 1) (S_Edit2_1))  ;[OK] ボタンが押された時のアクション
(princ)
);end
```



```
(setq help-S_Edit2  ; サブ関数 (help00) が呼び出された時 (alert 関数 ) に、その中に表示する文字列を記述する
  (strcat
    "¥n 面取りのタイプを選択します。"  ;¥n は改行
    "¥n  "
    "¥n 左側は、2 つの面取り距離を指定するタイプです。"
    "¥n 右側は、面取り距離と角度を指定するタイプです。"
    "¥n  "
    "¥nOK ボタンを押してを処理を継続してください。"
  )
)
```


```
;<< サブ関数の定義 >>
(defun help00()  ; サブ関数 (help00) が呼び出された時に、alert 関数を使用する
  (alert help-S_Edit2)  ;help-S_Edit2 の文字列を表示する
);end
;<< 補助関数の定義 >>
(defun S_Edit2_1()
  (cond
    ( (= check 1) (cham1) )  ; 変数 [check] の値が <1> のときは、補助関数 (cham1) を起動する
    ( (= check 2) (cham2) )  ; 変数 [check] の値が <2> のときは、補助関数 (cham2) を起動する
    ( T           (cham1) )  ; どちらでも無い場合は、補助関数 (cham1) を起動する
  )
);end
;<< 外部サブ関数の定義 >>
(defun cham1()
  (setvar "CHAMMODE" 0)  ;2 つの面取り距離を指定する
  (setvar "CHAMFERA" (atof sen1))  ;1 本目の面取り距離を設定（文字列を実数に変換）
  (setvar "CHAMFERB" (atof sen2))  ;2 本目の面取り距離を設定（文字列を実数に変換）
  (prompt "¥n1 本目の線分と 2 本目の線分を選択：")  ; コマンドラインにメッセージを表示
  (command "CHAMFER")  ; 面取り [CHAMFER] を使って面取りを行う。2 本の線分はユーザーが指定する。
  (setq sen1 nil sen2 nil kaku nil)  ; グローバル変数を空 <nil> にセット
);end
;<< 外部サブ関数の定義 >>
(defun cham2()
  (setvar "CHAMMODE" 1)  ; 面取り距離と角度を指定する
  (setvar "CHAMFERC" (atof sen1))  ; 面取りの長さを設定（文字列を実数に変換）
  (setvar "CHAMFERD" (angtof kaku 0))  ; 面取りの角度を設定（角度文字列を度数に変換）
  (prompt "¥n1 本目の線分と 2 本目の線分を選択：")  ; コマンドラインにメッセージを表示
  (command "CHAMFER")  ; 面取り [CHAMFER] を使って面取りを行う。2 本の線分はユーザーが指定する。
  (setq sen1 nil sen2 nil kaku nil)  ; グローバル変数を空 <nil> にセット
  );end
```

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	スライドファイルを表示する範囲（X と Y の座標値）を指定します。縦横の幅を dcl で設定した "image01" の数値を指定しています。
③	イメージの表示は、[start_image][slide_image][end_image] の 3 つを記述します。 [fill_image] を指定した場合は、イメージの背景色を指定できます。（<-2> はその時点の AutoCAD の背景色と同じになります。）
④	右側のイメージ (image02) も③と同様にします。
⑤	edit_box "sen1"、"sen2"、"kaku" に初期値を入力します。[set_tile] は [load_dialog] の後、[action_tile] の前に記述します。ここでは、""（空白）を指定していますが、既定値を指定することも出来ます。
⑥	edit_box "sen1" の内容を変数 "sen1" に代入します。（文字列）
⑦	edit_box "sen2" の内容を変数 "sen2" に代入します。（文字列）
⑧	edit_box "kaku" の内容を変数 "kaku" に代入します。（文字列）
⑨	icon_image "image01" が選択されると、プログラム分岐変数 check に <1> をセットします。 また edit_box "kaku" の mode_tile を <1> をセットすることで、"kaku" には入力が出来なくなります。
⑩	icon_image "image02" が選択されると、プログラム分岐変数 check に <2> をセットします。 また edit_box "sen2" の mode_tile を <1> をセットすることで、"sen2" には入力が出来なくなります。
⑪	"help" ボタンを押したときに、内部サブ関数 (help00) を起動します。
⑫	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑬	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_Edit2_1) を起動し、ダイアログを終了します。

Point!

mode_tile の値には、以下の 5 つのパターンがあります。

mode_tile の mode のコード	
値	説　明
0	タイルを使用可能にします。
1	タイルを使用禁止にします。
2	タイルにフォーカスを設定します。
3	編集ボックスの内容を選択します。
4	イメージのハイライト表示のオンとオフを切り替えます。

```
(action_tile "image01"                  ; "image01" が選択された場合
  (strcat
    "(setq check 1)"                    ; 分岐変数 [check] に <1> をセット
    "(mode_tile ¥"kaku¥" 1)"            ; タイル "kaku" を使用禁止にします。
    "(mode_tile ¥"sen2¥" 0)"            ; タイル "sen2" を使用可能にします。
  )
)
```

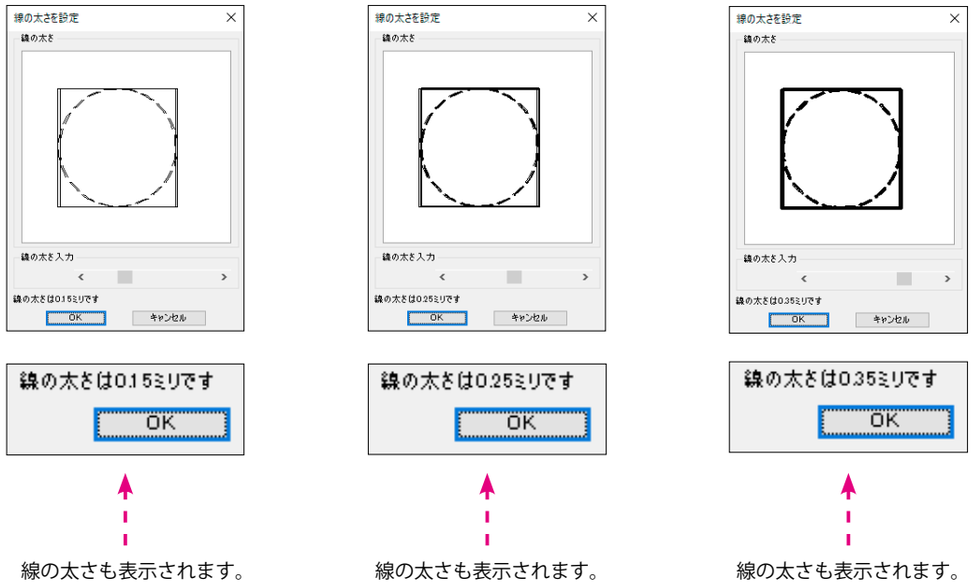
[mode_tile] を使用すると、上記のように他のタイルの mode（使用禁止や使用可能）をコントロールできます。

第 6 節　スライダーをコントロールする

使用する LISP	S_Slider1.lsp	使用する主な関数	setvar
内部サブ関数	(GetTileVal)	外部サブ関数	(S_Slider1_1)
使用する DCL	S_Slider1.dcl	使用するタイル	icon_image、slider、text
外部テキスト	なし	スライドファイル	S_Slider1-1 ～ S_Slider1-5
このプログラムは、線の太さの設定を画像で確認しながら行うことができます。 スライダーを動かすたびに、画像の線の太さが変わりますから、視覚的に判りやすくなります。			

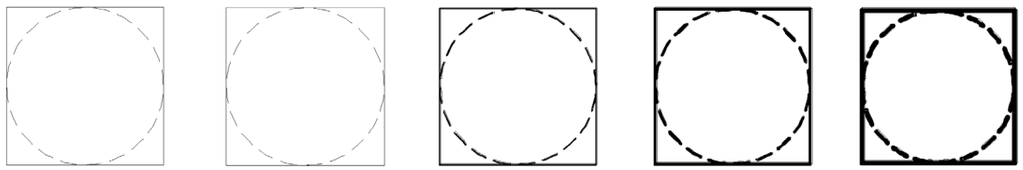
Step1 － <S_Slider1.lsp> を起動すると、下図のダイアログが表示されます。
スライダーを動かすと、画像の線の太さが変更されます。
また、スライダーの下のテキストボックスに線の太さの値も表示されます。

Step2 －スライダーで線の太さを確認しながら、新しい線の太さを指定できます。



線の太さも表示されます。　線の太さも表示されます。　線の太さも表示されます。

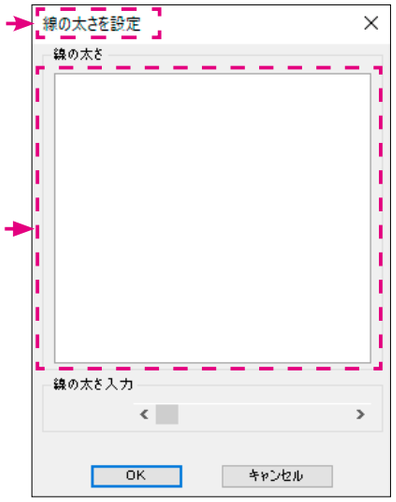
5 つのスライドファイル (sld) は事前に用意しておきます。

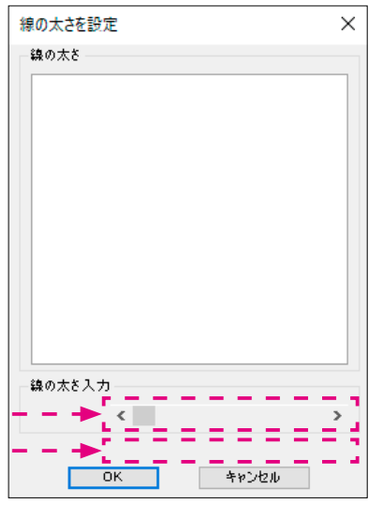


S_Slider1-1.sld　S_Slider1-2.sld　S_Slider1-3.sld　S_Slider1-4.sld　S_Slider1-5.sld

使用するダイアログ	S_Slider1.dcl

```
S_Slider1 : dialog
{
  label = " 線の太さを設定 ";

    :boxed_column /* タイルを縦並びに配置スタート */
    {
      label = " 線の太さ ";

      :icon_image /* イメージ処理タイルスタート */
      {
        label = " パターン図 ";
        key = "image00";
        width = 40;
        fixed_width = true;
        height =20;
      }       /* イメージ処理タイル終了 */
    }       /* タイルを縦並びに配置終了 */
    :boxed_row /* タイルを横並びに配置スタート */
    {
      label = " 線の太さ入力 ";
       spacer;
       :slider       /* スライダー処理タイルスタート */
       {
          key = "slider01";
          min_value = 5;
          max_value = 45;
          small_increment = 10;
          big_increment = 10;
          width = 30;
          fixed_width = true;
          is_tab_stop = false;
       }       /* スライダー処理タイル終了 */
    }       /* タイルを横並びに配置終了 */

    :text /* テキストボックス処理タイルスタート */
     {
        key = "text00";
        width = 20;
     }       /* テキストボックス処理タイル終了 */
  /* OK or CANCEL */
   ok_cancel;
}//end
```





タイルをコントロール

使用する LISP	S_Slider1.lsp

```
(defun c:S_Slider1( / dialog_box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Slider1"))   ;S_Slider1.dcl をメモリーに読み込む
  (new_dialog "S_Slider1" dcl_id)           ;S_Slider1 ダイアログを初期化する
                                            → ダイアログの読み込みと表示

;<< イメージを表示する範囲を指定 >>
❷ (setq x1 0) (setq y1 0)                   ; スライドを表示する位置（原点は左上）
  (setq x2 (dimx_tile "image00")            ; スライドを表示する X の幅（右方向）
        y2 (dimy_tile "image00")            ; スライドを表示する Y の幅（下方向）
  )
                                            → スライドを表示する X と Y の幅として "image00" の幅を指定しています。
;<< スライドを表示する準備 >>
❸ (start_image "image00")                   ;"image00" にスライドを読み込む
  (fill_image x1 y1 x2 y2 -2)               ; 背景色を現在の AutoCAD と同色
  (slide_image x1 y1 x2 y2 "S_Slider1-1")   ; スライド (S_Slider1-1) を表示
  (end_image)                               ; スライドの読み込みを終了
                                            → スライドを読み込み、背景色を指定して、"S_Slider1-1" のスライドを指定します。
❹ (action_tile "slider01" "(GetTileVal)")   ; "slider01" のバーがスライドした時、(GetTileVal) を起動
;<<"slider01" の切り替えを行うサブ関数 >>
❺ (defun GetTileVal()
  (setq p (get_tile "slider01"))    ;"slider01" の値を変数 <p> にセット
  (setq p1 (rtos (/ (atof p) 100)))  ;<p1> の値は 1/100 ミリメートル単位で入力する必要があります
  (setq pt0 " 線の太さは ")  ;" 線の太さは " の文字を <pt0> にセット
  (set_tile "text00" (strcat pt0  p1 " ミリです "))  ;text00 に 文字列を連結してセット
  (cond ((= p "5")(setq image000 "S_Slider1-1"))  ;p が <"5"> の時、image000 に "S_Slider1-1" を代入
        ((= p "15")(setq image000 "S_Slider1-2")) ;p が <"15"> の時、image000 に "S_Slider1-2" を代入
        ((= p "25")(setq image000 "S_Slider1-3")) ;p が <"25"> の時、image000 に "S_Slider1-3" を代入
        ((= p "35")(setq image000 "S_Slider1-4")) ;p が <"35"> の時、image000 に "S_Slider1-4" を代入
        ((= p "45")(setq image000 "S_Slider1-5")) ;p が <"45"> の時、image000 に "S_Slider1-5" を代入
  )
  (start_image "image00")                   ;"image00" にスライドを読み込む
  (fill_image x1 y1 x2 y2 -2)               ; 背景色を現在の AutoCAD と同色にする
  (slide_image x1 y1 x2 y2 image000)        ; スライド (image000) を表示
  (end_image)                               ; スライドの読み込みを終了
                                            → スライドを読み込み、背景色を指定して、"image000" のスライドを指定します。
)
```

(image000) には、"S_Slider1-1" から "S_Slider1-5" のスライドが順次切り替わります。

タイルをコントロール

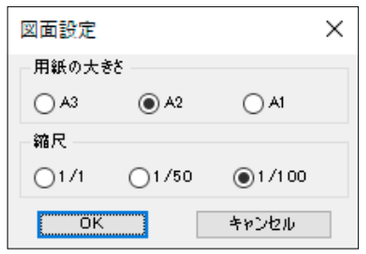
```
⑥  (action_tile "accept" "(S_Slider1_1)(done_dialog 1)") ; ダイアログを閉じて (S_Slider1_1) を起動
   (action_tile "cancel" "(done_dialog 0)")              ; 何も処理しないで閉じる
  ;<< ダイアログを終了する準備 >>
⑦  (setq dialog-box (start_dialog))   ; ユーザー入力を受け付ける
   (unload_dialog dcl_id)             ; ダイアログをメモリーから解放する
                                                  ]→ ダイアログの入力と終了
  (princ)
 );end
 ;<< 外部サブ関数に移る >>
⑧ (defun S_Slider1_1()
  (cond ; スライダーの値が <1 ～ 5> であれば、[ 線の太さを表示 <LWDISPLAY>] を有効にして、
 ;<< 既定の線の太さの値を <0.05> に設定します。>>
    ( (= p "1") (setvar "LWDISPLAY" 1)(setvar "LWDEFAULT" 5))        ;1/100 ミリメートル単位で入力
 ;<< 既定の線の太さの値を <0.15> に設定します。>>
    ( (= p "2") (setvar "LWDISPLAY" 1)(setvar "LWDEFAULT" 15))       ;1/100 ミリメートル単位で入力
 ;<< 既定の線の太さの値を <0.25> に設定します。>>
    ( (= p "3") (setvar "LWDISPLAY" 1)(setvar "LWDEFAULT" 25))       ;1/100 ミリメートル単位で入力
 ;<< 既定の線の太さの値を <0.35> に設定します。>>
    ( (= p "4") (setvar "LWDISPLAY" 1)(setvar "LWDEFAULT" 35))       ;1/100 ミリメートル単位で入力
 ;<< 既定の線の太さの値を <0.5> に設定します。>>
    ( (= p "5") (setvar "LWDISPLAY" 1)(setvar "LWDEFAULT" 50))       ;1/100 ミリメートル単位で入力
  );cond
 );end
```

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	スライドファイルを表示する範囲（X と Y の座標値）を指定します。縦横の幅を dcl で設定した "image00" の数値を指定しています。
③	イメージの表示は、[start_image][slide_image][end_image] の 3 つを記述します。 [fill_image] を指定した場合は、イメージの背景色を指定できます。（<-2> はその時点の AutoCAD の背景色と同じになります。）
④	"slider01" が操作される度に、内部サブ関数 (SaveTileVal) を起動します。
⑤	"slider01" の値を変数 [p] にセットします。p の値に応じて、スライドファイルの表示を切り替えます。 p の値は文字なので、ミリ単位で表示するために数値に変換します。 （既定の線の太さは、太さの値を (1/100 ミリメートル単位) で設定します。） text_tile "text00" に現在の線の太さを表示します。 p の値に応じて、対応するスライドファイルを切り替えて表示します。
⑥	[OK] ボタンが押されると、外部サブ関数 (S_Slider1_1) を起動して、フラッグ (status) として <1> を指定します。 この数値はユーザーが指定できます。
⑦	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑧	"slider01" の値が変更される度に、システム変数 [LWDISPLAY] の値を変更します。 [LWDISPLAY] は線の太さをコントロールするシステム変数です。

第 7 節　図枠を自動的に作成する

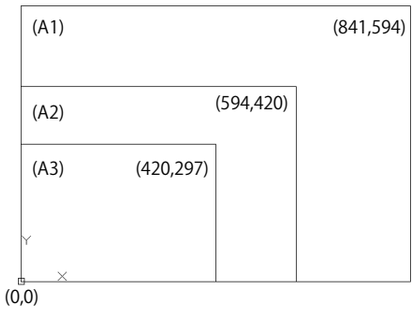
使用する LISP	S_Radio1.lsp	使用する主な関数	AutoCAD("RECTANG"、"ZOOM")、if
内部サブ関数	なし	外部サブ関数	(S_Radio1_1)、(check_9) ～ (check36)
使用する DCL	S_Radio1.dcl	使用するタイル	Radio_button
外部テキスト	なし	スライドファイル	なし
このプログラムは、用紙の大きさと尺度を指定して、自動的に図枠を作図します。 定型の図枠をまとめて指定しておけば、効率が上がります。			

Step1 ー <S_Radio1.lsp> を起動すると、下図のダイアログが表示されます。
用紙の大きさと縮尺を選択して、[OK] ボタンを押します。
指定しない場合は、用紙は [A2]、縮尺は [1/100] になります。

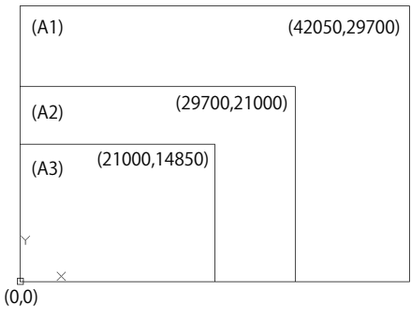


Step2 ー図枠の左下は、原点 (0,0) を既定値として指定しています。
第 4 節の方法を取り入れると、挿入時の位置をマウスで指定することが可能です。

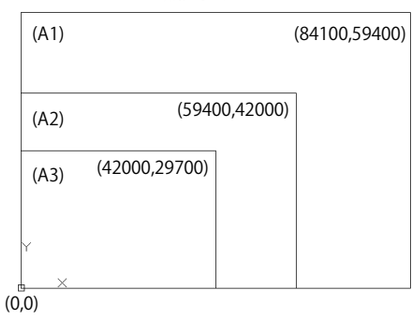
縮尺 1/1



縮尺 1/50

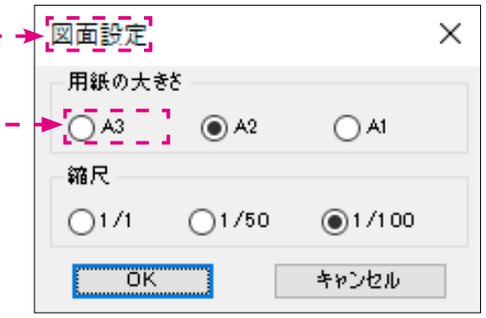


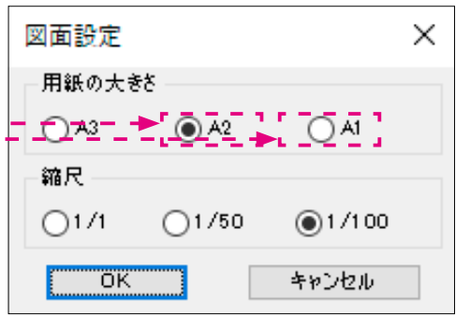
縮尺 1/100



使用するダイアログ	S_Radio1.dcl

```
S_Radio1 : dialog
{
  label = " 図面設定 ";
  :boxed_radio_row  /* タイルを横並びに配置スタート */
  {
    label = " 用紙の大きさ ";
    :radio_button
    {
       label = "A3";
       key = "A3";
    }
    :radio_button
    {
       label = "A2";
       key = "A2";
       value = "1";
    }
    :radio_button
    {
       label = "A1";
       key = "A1";
    }
  }    /* タイルを横並びに配置終了 */
```





Point!

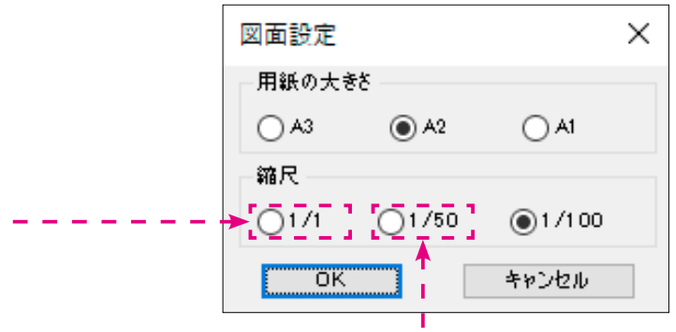
radio_button の値

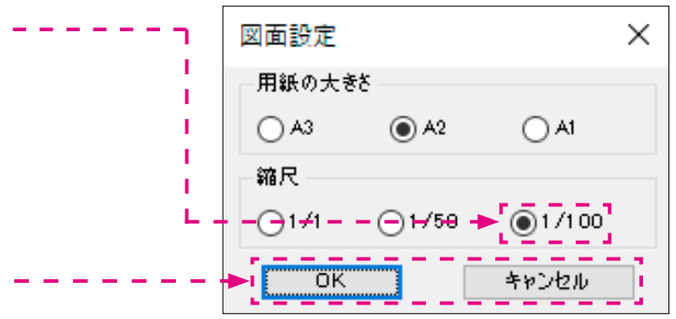
radio_button が取得する値は、<"0"> か <"1"> のどちらかです。
選択された場合は <"1"> がセットされ、選択されない場合は <"0"> がセットされます。
toggle_button と違って、radio_button はグループの内、1 つだけがオンに出来ます。
例えば、上記の中で key="A2" が選択された時（オン）、他の 2 つ（"A3" と "A1"）は自動的にオフになります。

label が "A2" の radio_button が選択された時、それぞれ 3 つのボタンの値は次のようになります。

- key = "A3" の値は、(setq a3 (get_tile "A3")) で取得できます。
  このとき、a3 には <"0"> がセットされています。（文字列）
- key = "A2" の値は、(setq a2 (get_tile "A2")) で取得できます。
  このとき、a2 には <"1"> がセットされています。（文字列）
- key = "A1" の値は、(setq a1(get_tile "A1")) で取得できます。
  このとき、a1 には <"0"> がセットされています。（文字列）


```
:boxed_row    /* タイルを横並びに配置スタート */
{
  label = " 縮尺 ";

  :radio_button
  {
     label = "1/1";
     key = "1/1";
  }

  :radio_button
  {
     label = "1/50";
     key = "1/50";
  }

  :radio_button
  {
     label = "1/100";
     key = "1/100";
     value = "1";
  }
}    /* タイルを横並びに配置終了 */

 /* OK or CANCEL */
   ok_cancel;
}//end
```





Point!

radio_button [key = "1/100"] に初期値をセットする方法

ダイアログ側では、[value] に <"1"> を記述します。

```
:radio_button
    {
       label = "1/100";
       key = "1/100";
       value = "1";
    }
```

AutoLISP 側では、[set_tile] で <"1"> を代入します。

```
(new_dialog "S_Radio1" dcl_id)
      .
;new_dialog と start_dialog の間に記述する
(set_tile "1/100" "1")
      .
(setq dialog-box (start_dialog))
```


タイルをコントロール

使用する LISP	S_Radio1.lsp

```
(defun c:S_Radio1( / dialog_box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Radio1"))   ;S_Radio1.dcl をメモリーに読み込む
  (new_dialog "S_Radio1" dcl_id)           ;S_Radio1 ダイアログを初期化する
                                           → ダイアログの読み込みと表示

;<< 用紙の大きさのチェック >>
❷ (action_tile "A3" "(setq yousi 1)")      ; "A3" が選択されると、<1> をセット
  (action_tile "A2" "(setq yousi 2)")      ; "A2" が選択されると、<2> をセット
  (action_tile "A1" "(setq yousi 4)")      ; "A1" が選択されると、<4> をセット
    (if (= yousi nil)                      ; どれも選択されなかった時の処理
      (setq yousi 2)                       ; <2> をセット
    )                                      → どれも選択されなかった時は、変数 [yousi] に <2> をセットします。

;<< 縮尺のチェック >>
❸ (action_tile "1/1" "(setq scale 8)")     ; "1/1" が選択されると、<8> をセット
  (action_tile "1/50" "(setq scale 16)")   ; "1/50" が選択されると、<16> をセット
  (action_tile "1/100" "(setq scale 32)")  ; "1/100" が選択されると、<32> をセット
    (if (= scale nil)                      ; どれも選択されなかった時の処理
      (setq scale 32)                      ; <32> をセット
    )                                      → どれも選択されなかった時は、変数 [scale] に <32> をセットします。

;<< ダイアログを終了する準備 >>
❹ (setq dialog-box (start_dialog))         ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                   ; ダイアログをメモリーから解放する
                                           → ダイアログの入力と終了

;<<[OK] ボタンが押されたら、外部サブ関数 (S_Radio1_1) を起動する >>
❺ (if(= (getvar "DIASTAT") 1) (S_Radio1_1))  ;[OK] ボタンが押された時、外部サブ関数 (S_Radio_1) を起動

(princ)
);end
```

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	選択した radio_button の値を取得します。"A3" は <1>、"A2" は <2>、"A1" は <4> にします。 どれも選択されなかった場合には、エラーを避けるために <2> をセットします。
③	②と同様に、"1/1" を <8>、"1/50" を <16>、"1/100" を <32> にセットします。 どれも選択されなかった場合には、エラーを避けるために <32> をセットします。
④	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑤	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_Radio1_1) を起動し、ダイアログを終了します。

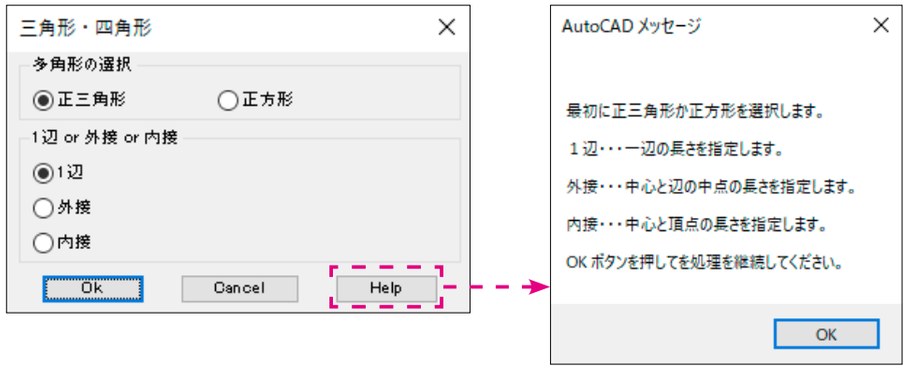

```
(defun S_Radio1_1()
  (setq check (+ yousi scale))  ; 変数 [yousi] と変数 [scale] を合計する
  (cond
    ( (= check 9)  (check_9) )          ; 合計 (check) が <9> の時は、外部サブ関数 (check_9) を起動
    ( (= check 17)  (check_17) )        ; 合計 (check) が <17> の時は、外部サブ関数 (check_17) を起動
    ( (= check 33)  (check_33) ); 以上 A3  ; 合計 (check) が <33> の時は、外部サブ関数 (check_33) を起動
    ( (= check 10)  (check_10) )        ; 合計 (check) が <10> の時は、外部サブ関数 (check_10) を起動
    ( (= check 18)  (check_18) )        ; 合計 (check) が <18> の時は、外部サブ関数 (check_18) を起動
    ( (= check 34)  (check_34) ); 以上 A2  ; 合計 (check) が <34> の時は、外部サブ関数 (check_34) を起動
    ( (= check 12)  (check_12) )        ; 合計 (check) が <12> の時は、外部サブ関数 (check_12) を起動
    ( (= check 20)  (check_20) )        ; 合計 (check) が <20> の時は、外部サブ関数 (check_20) を起動
    ( (= check 36)  (check_36) ); 以上 A1  ; 合計 (check) が <36> の時は、外部サブ関数 (check_36) を起動
  );cond
  (setq yousi nil scale nil)  ; 変数 [yousi] と変数 [scale] の値を <nil> にセット
(princ)
);end
;<<A3 用紙　縮尺 1/1>>
(defun check_9 ( )
  (command "rectang" "0,0" "420,297")       ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                      ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<<A3 用紙　縮尺 1/50>>
(defun check_17 ( )
  (command "rectang" "0,0" "21000,14850")   ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                      ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<<A3 用紙　縮尺 1/100>>
(defun check_33 ( )
  (command "rectang" "0,0" "42000,29700")   ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                      ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<< 以上 A3>>
```

```
;<<A2 用紙　縮尺 1/1>>
(defun check_10 ( )
  (command "rectang" "0,0" "594,420")          ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                         ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<<A2 用紙　縮尺 1/50>>
(defun check_18 ( )
  (command "rectang" "0,0" "29700,21000")      ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                         ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<<A2 用紙　縮尺 1/100>>
(defun check_34 ( )
  (command "rectang" "0,0" "59400,42000")      ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                         ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<< 以上 A2>>
;<<A1 用紙　縮尺 1/1>>
(defun check_12 ( )
  (command "rectang" "0,0" "841,594")          ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                         ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<<A1 用紙　縮尺 1/50>>
(defun check_20 ( )
  (command "rectang" "0,0" "42050,29700")      ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                         ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<<A1 用紙　縮尺 1/100>>
(defun check_36 ( )
  (command "rectang" "0,0" "84100,59400")      ; 長方形 [rectang] で作図する
  (command "ZOOM" "A")                         ; 拡大 [zoom] で図面全体を表示する
(princ)
);end
;<< 以上 A1>>
```

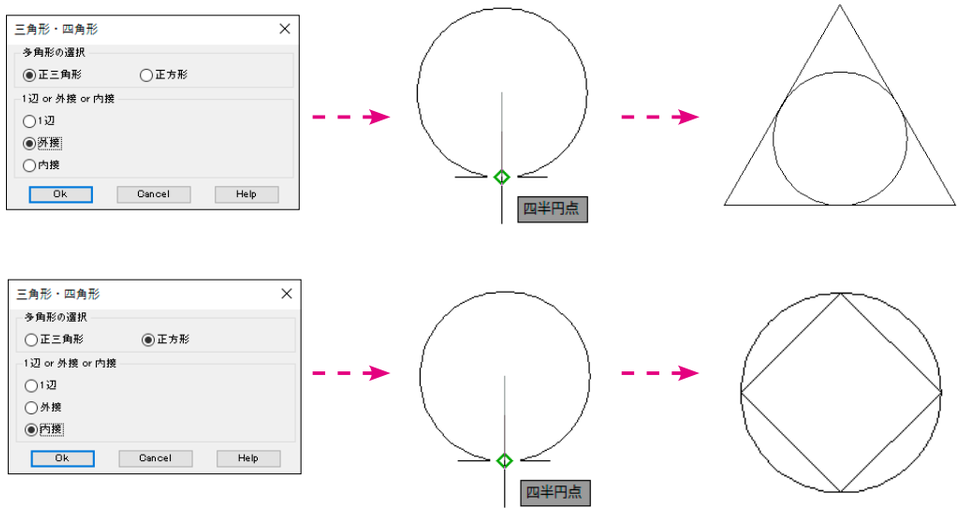

第 8 節　多角形を視覚的に作図する

使用する LISP	S_Radio2.lsp	使用する主な関数	cmdecho、AutoCAD("polygon")
内部サブ関数	(GetTileVal)	外部サブ関数	(S_Radio2_1)、(san1) 〜 (sikaku3)
使用する DCL	S_Radio2.dcl	使用するタイル	Radio_button
外部テキスト	なし	スライドファイル	なし
このプログラムは、多角形 (POLYGON) の作図方法を視覚的に判りやすく表示したものです。 コマンドラインに目を落とすことも無いので、作業が中断することはありません。			

Step1 ー <S_Radio2.lsp> を起動すると、下図のダイアログが表示されます。
最初に、正三角形か正方形かを選択します。次に、3 つの作図方法から 1 つを選択します。
[Help] ボタンを押すと、3 つの作図方法が表示されます。



Step2 ー上の図は、正三角形の [外接] を選択した図です。
下の図は、正方形の [内接] を選択した図です。

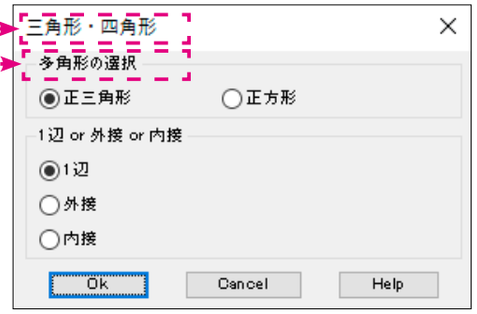


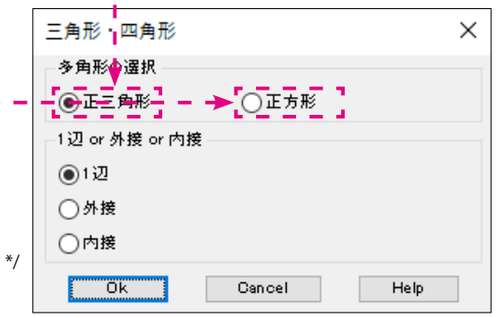
使用するダイアログ	S_Radio 2.dcl

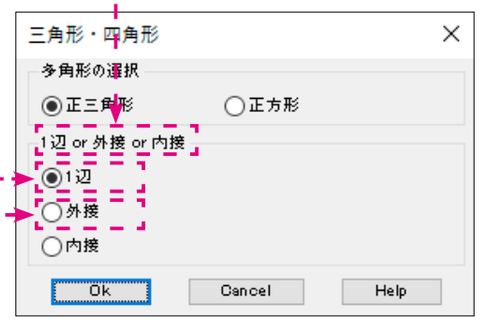
```
S_Radio2 : dialog
{
  label = " 三角形・四角形 ";
  :boxed_radio_row    /* タイルを横並びに配置スタート */
  {
    label = " 多角形の選択 ";
    key = "main_key1;
    :radio_button
    {
      label = " 正三角形 ";
      key = "sankaku";
    }
    :radio_button
    {
      label = " 正方形 ";
      key = "sikaku";
    }
    spacer;
  }    /* タイルを横並びに配置終了 */
  :boxed_radio_column    /* タイルを縦並びに配置スタート */
  {
    label = "1 辺 or 外接 or 内接 ";
    key = "main_key2
    :radio_button
    {
      label = "1 辺 ";
      key = "hen1";
    }
    :radio_button
    {
      label = " 外接 ";
      key = "hen2";
    }
```

Point!

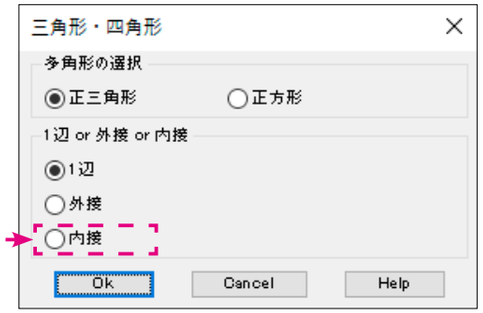
Point!

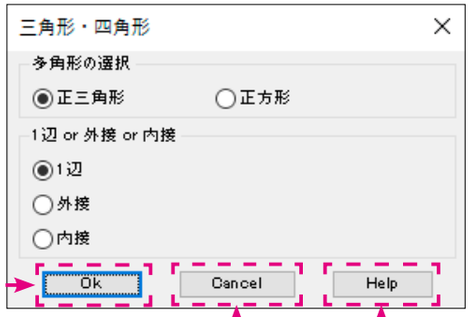






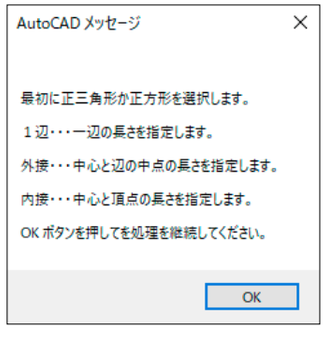

```
    :radio_button
    {
      label = " 内接 ";
      key = "hen3";
    }
  }    /* タイルを縦並びに配置終了 */

  /* OK or CANCEL */
  : row      /* タイルを横並びに配置スタート */
  {
    spacer;
    : button
    {
      label = "Ok";
      is_default = true;
      key = "accept";
      width = 8;
      fixed_width = true;
    }
    spacer;
    : button
    {
      label = "Cancel";
      is_cancel = true;
      key = "Cancel";
      width = 4;
      fixed_width = true;
    }
    spacer;
    : button
    {
      label = "Help";
      key = "help";
      width = 4;
      fixed_width = true;
    }
    spacer;
  }    /* タイルを横並びに配置終了 */
}//end
```





使用する LISP	S_Radio 2.lsp

```
(defun c:S_Radio2( / dialog_box dcl_id help-radio_2)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Radio2"))   ;S_Radio2.dcl をメモリーに読み込む
  (new_dialog "S_Radio2" dcl_id)           ;S_Radio2 ダイアログを初期化する
                                           → ダイアログの読み込みと表示
;<<radio_button[ 多角形の選択 ] のどちらが選択されたか、[1 辺 or 外接 or 内接 ] のどれが選択されたかを調べる >>
❷ (action_tile "main_key1" "(GetTileVal)")  ; "main_key1" に値が入ると、サブ関数 (GetTileVal) を起動
  (action_tile "main_key2" "(GetTileVal)")  ; "main_key2" に値が入ると、サブ関数 (GetTileVal) を起動

Point!
❸ (defun GetTileVal ()
    (setq a (get_tile "main_key1"))   ; "main_key1" 内の選択された項目 (<sankaku>or<sikaku>) を取得
    (setq b (get_tile "main_key2"))   ; "main_key2" 内の選択された項目 (<hen1>or<hen2>or<hen3>) を取得
  )
❹ (action_tile "help" "(help00)")     ; "help" ボタンが押された時、サブ関数 (help00) を起動
;<< ダイアログを終了する準備 >>
❺ (setq dialog-box (start_dialog))    ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)              ; ダイアログをメモリーから解放する
                                      → ダイアログの入力と終了
;<<[OK] ボタンが押されたら、外部サブ関数 (S_Radio2_1) を起動する >>
❻ (if(= (getvar "DIASTAT") 1) (S_Radio2_1))  ;[OK] ボタンが押された時のアクション
  (princ)
);end
;<< 内部サブ関数 >>
❼ (defun help00()                     ; 内部サブ関数 (help00) が呼び出された時に、alert 関数を使用する
    (alert help-radio_2)              ;[help-radio_2] のテキストを表示する
  )
;<< サブ関数 (help00) が呼び出された時 (alert 関数 ) に、その中に表示する文字列を記述する >>
(setq help-radio_2
  (strcat
    "¥n 最初に正三角形か正方形を選択します。"
    "¥n  "
    "¥n  1 辺・・・一辺の長さを指定します。"
    "¥n  "
    "¥n 外接・・・中心と辺の中点の長さを指定します。"
    "¥n  "
    "¥n 内接・・・中心と頂点の長さを指定します。"
    "¥n  "
    "¥nOK ボタンを押してを処理を継続してください。"
  ))
```




```
;<< 外部サブ関数 >>
❽ (defun S_Radio2_1( / c)

Point!
  (setq c (strcat a b))  ;A の文字列と B の文字列を結合した内容次第で外部サブ関数を分岐する
  (cond  ; 内容次第で外部サブ関数を分岐する
   ( (= c "sankakuhen1")  (san1) )  ;c の文字列が <sankakuhen1> の時は、外部サブ関数 (san1) を起動
   ( (= c "sankakuhen2") (san2) )  ;c の文字列が <sankakuhen2> の時は、外部サブ関数 (san2) を起動
   ( (= c "sankakuhen3") (san3) )  ;c の文字列が <sankakuhen3> の時は、外部サブ関数 (san3) を起動
   ( (= c "sikakuhen1") (sikaku1) )  ;c の文字列が <sikakuhen1> の時は、外部サブ関数 (sikaku1) を起動
   ( (= c "sikakuhen2") (sikaku2) )  ;c の文字列が <sikakuhen2> の時は、外部サブ関数 (sikaku2) を起動
   ( (= c "sikakuhen3") (sikaku3) )  ;c の文字列が <sikakuhen3> の時は、外部サブ関数 (sikaku3) を起動
   (   T          (san1) )  ; どれも選択されなかった時は、外部サブ関数 (san1) を起動
  );cond
 (princ)
 );end
;<<1 辺を指示して正三角形を作図 >>
(defun san1 ( / pt1 pt2)
  (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
  (setq pt1 (getpoint "¥n 辺の 1 点目 :"))  ; 辺の 1 点目を pt1 にセット
  (setq pt2 (getpoint pt1 "¥n 辺の 2 点目 :"))  ; 辺の 2 点目を pt2 にセット
  (command "polygon" "3" "E" pt1 pt2)  ;[ ポリゴン ] コマンドで正三角形を作図
  (setq  a  nil  b  nil)  ; グローバル変数を <nil> にセット
  (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
 );end
;<< 中心と辺の中点までの距離を指示して正三角形を作図 >>
 (defun san2 ( / pt1 pt2)
  (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
  (setq pt1 (getpoint "¥n 中心 :"))  ; 中心を pt1 にセット
  (setq pt2 (getpoint pt1 "¥n 辺の中点 :"))  ; 辺の中点までの距離を pt2 にセット
  (command "polygon" "3" pt1 "C" pt2)  ;[ ポリゴン ] コマンドで正三角形を作図
  (setq  a  nil  b  nil)  ; グローバル変数を <nil> にセット
  (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
 );end
```

Point!

❽ [多角形の選択] で < 三角形 > を選んだ場合は、変数 <a> には <sankaku> が代入されています。
[1 辺 or 外接 or 内接] で <1 辺 > を選んだ場合は、変数 <b> には <hen1> が代入されています。
この 2 つを文字列連結の関数 [strcat] で 1 つの文字列 <sankakuhen1> にしています。

```
;<< 中心と頂点までの距離を指示して正三角形を作図
(defun san3 ( / pt1 pt2)
 (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
 (setq pt1 (getpoint "¥n 中心 :"))  ; 中心を pt1 にセット
 (setq pt2 (getpoint pt1 "¥n 辺の頂点 :"))  ; 頂点までの距離を pt2 にセット
 (command "polygon" "3" pt1 "I" pt2)  ;[ ポリゴン ] コマンドで正三角形を作図
 (setq a nil b nil)  ; グローバル変数を <nil> にセット
 (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
);end
;<<1 辺を指示して正方形を作図 >>
;<< 四角形 >>
(defun sikaku1 ( / pt1 pt2)
 (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
 (setq pt1 (getpoint "¥n 辺の 1 点目 :"))  ; 辺の 1 点目を pt1 にセット
 (setq pt2 (getpoint pt1 "¥n 辺の 2 点目 :"))  ; 辺の 2 点目を pt2 にセット
 (command "polygon" "4" "E" pt1 pt2)  ;[ ポリゴン ] コマンドで正方形を作図
 (setq a nil b nil)  ; グローバル変数を <nil> にセット
 (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
);end
;<< 中心と辺の中点までの距離を指示して正方形を作図 >>
(defun sikaku2 ( / pt1 pt2)
 (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
 (setq pt1 (getpoint "¥n 中心 :"))  ; 中心を pt1 にセット
 (setq pt2 (getpoint pt1 "¥n 辺の中点 :"))  ; 辺の中点までの距離を pt2 にセット
 (command "polygon" "4" pt1 "C" pt2)  ;[ ポリゴン ] コマンドで正方形を作図
 (setq a nil b nil)  ; グローバル変数を <nil> にセット
 (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
);end
;<< 中心と頂点までの距離を指示して正方形を作図 >>
(defun sikaku3 ( / pt1 pt2)
 (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
 (setq pt1 (getpoint "¥n 中心 :"))  ; 中心を pt1 にセット
 (setq pt2 (getpoint pt1 "¥n 辺の頂点 :"))  ; 頂点までの距離を pt2 にセット
 (command "polygon" "4" pt1 "I" pt2)  ;[ ポリゴン ] コマンドで正方形を作図
 (setq a nil b nil)  ; グローバル変数を <nil> にセット
 (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
);end
```


番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	radio_button が選択される度に、内部サブ関数 (GetTileVal) を起動します。
③	2 つの radio_button のグループ値を取得します。("main_key1" と "main_key2") この 2 つの値は、それぞれの radio_button グループのキー名（文字列）が入ります。
④	"help" ボタンを押したときに、内部サブ関数 (help00) を起動します。
⑤	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑥	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_Radio2_1) を起動し、ダイアログを終了します。
⑦	[help] ボタンが押された時に、この関数を起動します。alert 関数で表示する文字列は、文字列変数名 (help-radio_2) に事前に記述しておきます。
⑧	[OK] ボタンが押された時に呼び出される外部関数です。

Point!

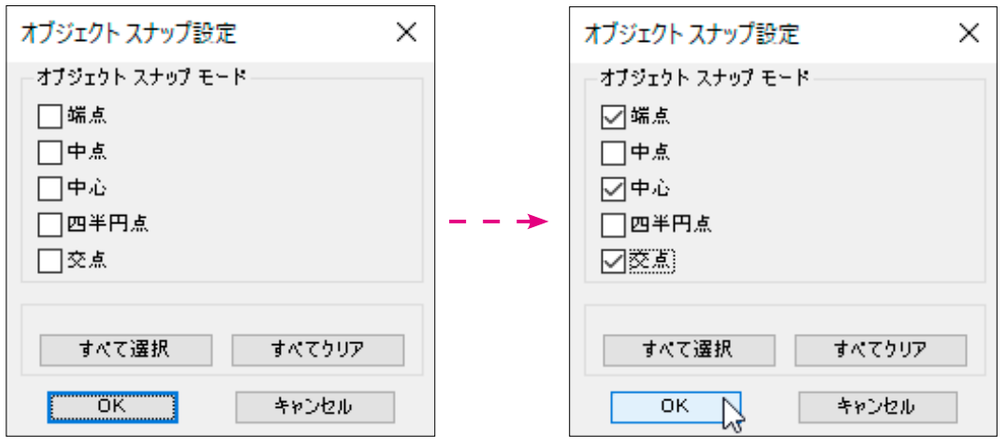
❸ Radio_button の [key] と Radio_row(column) の [key] の相違

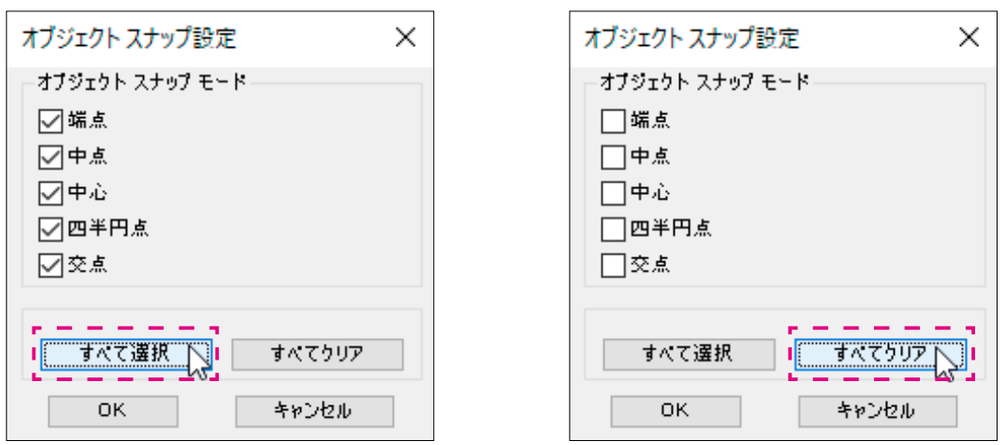
第 7 節では、radio_button の [key] の値を取得しました。radio_button の値は、<"0"> か <"1"> です。
第 8 節では、radio_row と radio_column の値を取得しました。
radio_row や radio_column が取得する値は、その radio グループを構成する各 radio_button の [key 名] になります。

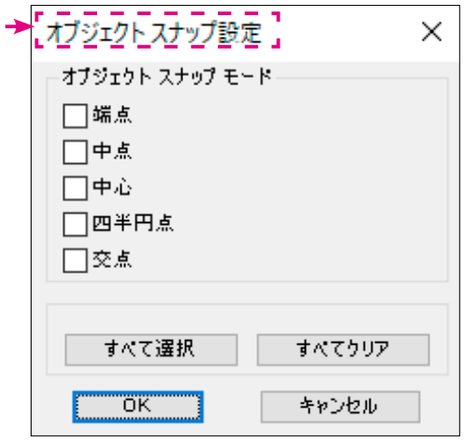
従って、[多角形の選択] のグループ（正三角形か正方形）の内、正三角形が選択された（"sankaku" が <"1"> にセット）されたときは、"main_key1" には "sankaku" の文字列がセットされます。
同様に、[1 辺 or 外接 or 内接] のグループの内、1 辺が選択された（"hen1" が <"1"> にセット）されたときは、"main_key2" には "hen1" の文字列がセットされます。

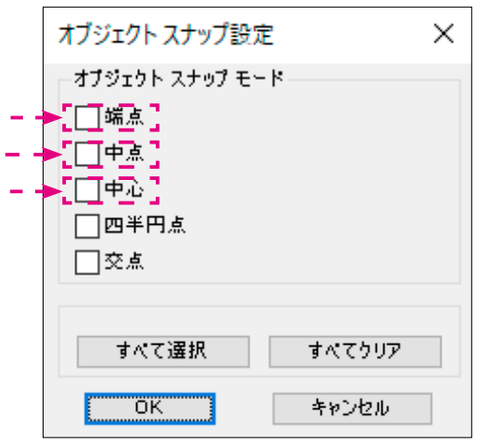
このように、radio_row や radio_column が取得する値は <"0"> や <"1"> ではなく、
構成する radio_button の [key 名] です。

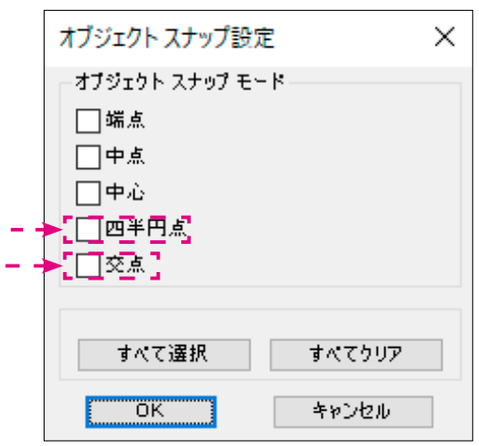
第 7 節では、radio_button の値（<"0"> か <"1"> か）でプログラムを分岐しましたが、
第 8 節では、radio_row や radio_column の値（具体的な <key 名 >）でプログラムを分岐しています。











第 9 節　ユーザー独自の O スナップ機能を作成する

使用する LISP	S_Toggle1.lsp	使用する主な関数	atoi、setvar
内部サブ関数	(S_Toggle1_Val) (osnap_all_set)、(osnap_all_clear)	外部サブ関数	(snap_val)
使用する DCL	S_Toggle1.dcl	使用するタイル	toggle
外部テキスト	なし	スライドファイル	なし
このプログラムは、AutoCAD の [オブジェクト スナップ] と同じ機能です。 このプログラムを作成すると、他のプログラムを起動中でも割り込みで使用出来ます。			

Step1 ー <S_Toggle1.lsp> を起動すると、下図のダイアログが表示されます。
この時点では、現在の [オブジェクト スナップ] の状態が表示されます。（左図）
使用するスナップを選択すると（右図）、AutoCAD の設定も自動的に変更されます。

オブジェクト スナップ設定
オブジェクト スナップ モード
☐端点
☐中点
☐中心
☐四半円点
☐交点

オブジェクト スナップ設定
オブジェクト スナップ モード
☑端点
☐中点
☑中心
☐四半円点
☑交点

Step2 ー [すべて選択] を選ぶと、5 つの項目すべてがチェックされます。（左図）
[すべてクリア] を選ぶと、5 つの項目すべてのチェックが解除されます。（右図）

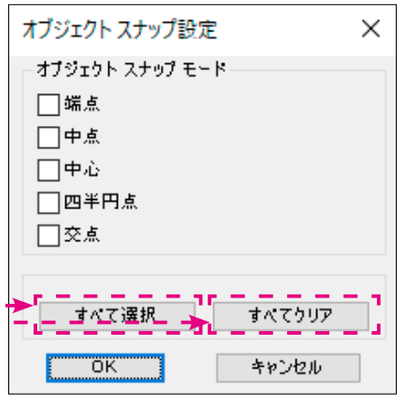
オブジェクト スナップ設定
オブジェクト スナップ モード
☑端点
☑中点
☑中心
☑四半円点
☑交点

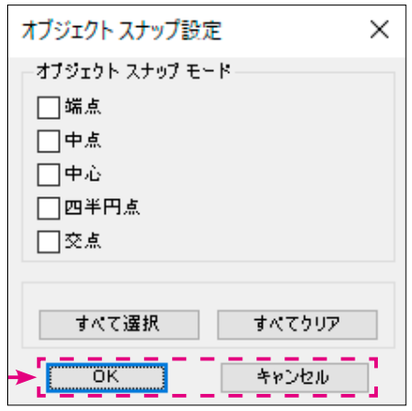
オブジェクト スナップ設定
オブジェクト スナップ モード
☐端点
☐中点
☐中心
☐四半円点
☐交点

使用するダイアログ	S_Toggle1.dcl

```
S_Toggle1 : dialog
{
 label = " オブジェクト スナップ設定 ";

   :boxed_column  /* タイルを縦並びに配置スタート */
   {
    label = " オブジェクト スナップ モード ";

     :toggle
     {
       label = " 端点 ";
       key = "snap_1";
     }

     :toggle
     {
       label = " 中点 ";
       key = "snap_2";
     }
     :toggle
     {
       label = " 中心 ";
       key = "snap_4";
     }
     :toggle
     {
       label = " 四半円点 ";
       key = "snap_16";
     }
     :toggle
     {
       label = " 交点 ";
       key = "snap_32";
     }
  }  /* タイルを縦並びに配置終了 */
```

```
:boxed_radio_row  /* タイルを横並びに配置スタート */
{
   label = "";
   :button
   {
      label = " すべて選択 ";
      key = "snap_all";
   }
   :button
   {
      label = " すべてクリア ";
      key = "snap_clear";
   }
 }    /* タイルを横並びに配置終了 */
 /* OK or CANCEL */
         ok_cancel;
}//end
```





Point!

osmode にセットする数値コード	
ビットコード	モード
0（※）	解除 [NON]（モード設定なし）
1	端点 [ENDpoint]
2	中点 [MIDpoint]
4	中心 [CENter]
8	点 [NODe]
16	四半円点 [QUAdrant]
32	交点 [INTersection]
64	挿入基点 [INSert]
128	垂線 [PERpendicular]
256	接線 [TANgent]
512	近接点 [NEArest]
1024（※）	すべてのオブジェクト スナップがクリアされます。
2048	仮想交点 [APP]
4096	延長 [EXT]
8192	平行 [PAR]

オブジェクト スナップを複数指定する場合は、これらの値の和を入力します。
たとえば、20 と入力すると、[中心](ビットコード <4>) と [四半円点](ビットコード <16>) のオブジェクト スナップを指定したことになります。

（※）(setvar "OSMODE" 0) と (setvar "OSMODE" 1024) は同じ結果になります。

使用する LISP	S_Toggle1.lsp

```
(defun c:S_Toggle1 ( / dialog_box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Toggle1"))   ;S_Toggle1.dcl をメモリーに読み込む
   (new_dialog "S_Toggle1" dcl_id)          ;S_Toggle1 ダイアログを初期化する
                                            → ダイアログの読み込みと表示
; <<"snap_1" から "snap_32" までの toggle が選択された時の動作を記述 >>
❷ (action_tile "snap_1" "(S_Toggle1_Val)")  ; "snap_1" が選択されると、サブ関数 (S_Toggle1_Val) を起動
   (action_tile "snap_2" "(S_Toggle1_Val)")  ; "snap_2" が選択されると、サブ関数 (S_Toggle1_Val) を起動
   (action_tile "snap_4" "(S_Toggle1_Val)")  ; "snap_4" が選択されると、サブ関数 (S_Toggle1_Val) を起動
   (action_tile "snap_16" "(S_Toggle1_Val)") ; "snap_16" が選択されると、サブ関数 (S_Toggle1_Val) を起動
   (action_tile "snap_32" "(S_Toggle1_Val)") ; "snap_32" が選択されると、サブ関数 (S_Toggle1_Val) を起動
; <<"snap_all" または、"snap_clear" が選択された時の動作を記述 >>
❸ (action_tile "snap_all"    "(osnap_all_sel)")   ; サブ関数 (osnap_all_sel) を起動
   (action_tile "snap_clear" "(osnap_all_clear)") ; サブ関数 (osnap_all_clear) を起動
;<< ダイアログを終了する準備 >>
❹ (setq dialog-box (start_dialog))  ; ユーザー入力を受け付ける
   (unload_dialog dcl_id)           ; ダイアログをメモリーから解放する
                                    → ダイアログの入力と終了
;<<[OK] ボタンが押されたら、外部サブ関数 (snap_val) を起動する >>
❺ (if(= (getvar "DIASTAT") 1) (snap_val)) ; システム変数 [DIASTAT] が <1> の時は、外部サブ関数を起動する
   (princ)
);end

❻ (defun S_Toggle1_Val() ; 各トグルの数値を変数にセットする (<"0"> か <"1">)
   (setq sna_1 (atoi (get_tile "snap_1")))   ;[snap_1] の情報 <"0" か "1"> を整数に変換して、変数にセット
   (setq sna_2 (atoi (get_tile "snap_2")))   ;[snap_2] の情報 <"0" か "1"> を整数に変換して、変数にセット
   (setq sna_4 (atoi (get_tile "snap_4")))   ;[snap_4] の情報 <"0" か "1"> を整数に変換して、変数にセット
   (setq sna_16 (atoi (get_tile "snap_16"))) ;[snap_16] の情報 <"0" か "1"> を整数に変換して、変数にセット
   (setq sna_32 (atoi (get_tile "snap_32"))) ;[snap_32] の情報 <"0" か "1"> を整数に変換して、変数にセット

   (if (= sna_1 1)(setq sna_1 1))   ;[sna_1] の値が <1> であれば、変数 [sna_1] に <1> をセット [ 端点 ]
   (if (= sna_2 1)(setq sna_2 2))   ;[sna_2] の値が <1> であれば、変数 [sna_2] に <2> をセット [ 中点 ]
   (if (= sna_4 1)(setq sna_4 4))   ;[sna_4] の値が <1> であれば、変数 [sna_4] に <4> をセット [ 中心 ]
   (if (= sna_16 1)(setq sna_16 16))  ;[sna_16] の値が <1> であれば、変数 [sna_16] に <16> をセット [ 四半円点 ]
   (if (= sna_32 1)(setq sna_32 32))  ;[sna_32] の値が <1> であれば、変数 [sna_32] に <32> をセット [ 交点 ]
);end
```

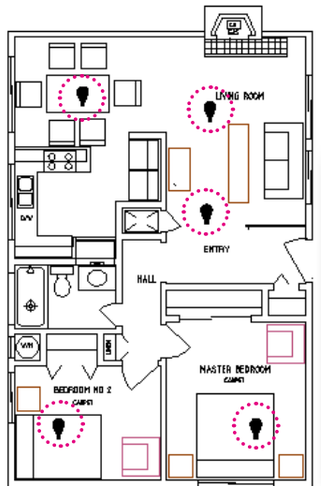
```
❼ (defun osnap_all_sel() ;[ すべて選択 ] ボタンが押された時の動作を記述
   (set_tile "snap_1" "1")   ;"snap_1" の値を <"1"(On)> にセット [ 端点 ]
   (set_tile "snap_2" "1")   ;"snap_2" の値を <"1"(On)> にセット [ 中点 ]
   (set_tile "snap_4" "1")   ;"snap_4" の値を <"1"(On)> にセット [ 中心 ]
   (set_tile "snap_16" "1") ;"snap_16" の値を <"1"(On)> にセット [ 四半円点 ]
   (set_tile "snap_32" "1") ;"snap_32" の値を <"1"(On)> にセット [ 交点 ]
❽ (S_Toggle1_Val) ; サブ関数 (S_Toggle1_Val) を起動
  );end

❾ (defun osnap_all_clear() ;[ すべてクリア ] ボタンが押された時の動作を記述
   (set_tile "snap_1" "0")    ;"snap_1" の値を <"0"(Off)> にセット
   (set_tile "snap_2" "0")    ;"snap_2" の値を <"0"(Off)> にセット
   (set_tile "snap_4" "0")    ;"snap_4" の値を <"0"(Off)> にセット
   (set_tile "snap_16" "0")   ;"snap_16" の値を <"0"(Off)> にセット
   (set_tile "snap_32" "0")   ;"snap_32 の値を <"0"(Off)> にセット
❿ (S_Toggle1_Val)           ; サブ関数 (S_Toggle1_Val) を起動
  );end
  ; 最終的な各トグルの数値を合計する
⓫ (defun snap_val ()
   (setvar "cmdecho" 0)  ; プロンプトとユーザー入力のエコーバックを非表示
   (setq sna_all (+ sna_1 sna_2 sna_4 sna_16 sna_32))  ; 選択された [snap] を合計
   (setvar "OSMODE" sna_all)  ; システム変数 [OSMODE] を設定
   (setq sna_1 nil sna_2 nil sna_4 nil sna_16 nil sna_32 nil sna_all nil)  ; 変数を <nil> にセット
   (setvar "cmdecho" 1)  ; プロンプトとユーザー入力のエコーバックを表示
  );end
```

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	toggle_button が選択される度に、内部サブ関数 (S_Toggle_Val) を起動します。
③	"snap_all" が選択された時は、内部サブ関数 (snap_all_sel) を起動します。 "snap_clear" が選択された時は、内部サブ関数 (snap_all_clear) を起動します。
④	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑤	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (snap_val) を起動し、ダイアログを終了します。
⑥	選択された toggle_button の値を整数に変換して変数にセットします。値は OSNAP のコードと同じです。 例えば、[中心] の OSNAP コードは <4> ですから、"snap_4" がチェックされたときは、変数 [sna-4] に <4> をセットしています。
⑦	[すべて選択] ボタンが押された時は、すべての toggle_button の値に <"1"> をセットします。
⑧	内部サブ関数 (S_Toggle_Val) で再計算して、総合計の値を計算します。
⑨	[すべてクリア] ボタンが押された時は、すべての toggle_button の値に <"0"> をセットします。
⑩	内部サブ関数 (S_Toggle_Val) で再計算して、総合計の値を計算します。
⑪	最終的な OSNAP の合計値を計算して、システム変数 [OSMODE] で O スナップ モードを再設定します。

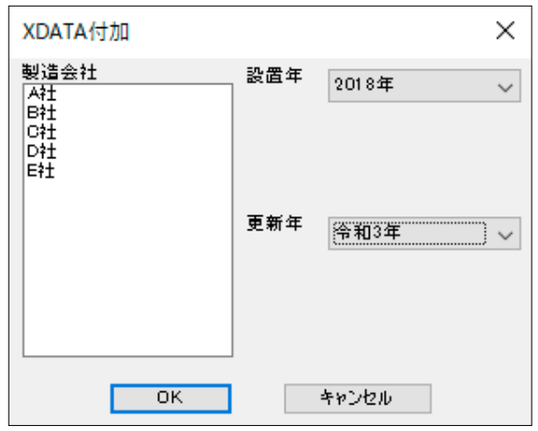
第 10 節　拡張データ (XDATA) を付加する

使用する LISP	S_Popup2.lsp	使用する主な関数	list、nth、mapcar、regapp、entmod
内部サブ関数	(SaveTileVal)	外部サブ関数	(S_Popup2_1)
使用する DCL	S_Popup2.dcl	使用するタイル	list_box、popup_list
外部テキスト	なし	スライドファイル	なし
このプログラムは、図面内の図形に拡張データ (Xdata) を付加します。 ブロックの属性定義と違って、通常の図形にユーザー独自のデータを付加することができます。			

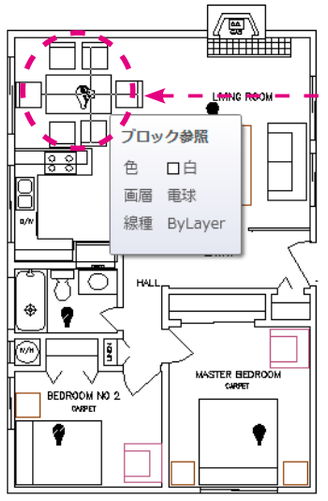
Step1 ー <S_Popup2.lsp> を起動すると、右下図のダイアログが表示されます。
[製造会社]、[設置年]、[更新年] を選択して、左図の中の電球を指示します。
ブロック図形以外の図形にも情報を付加することができます。

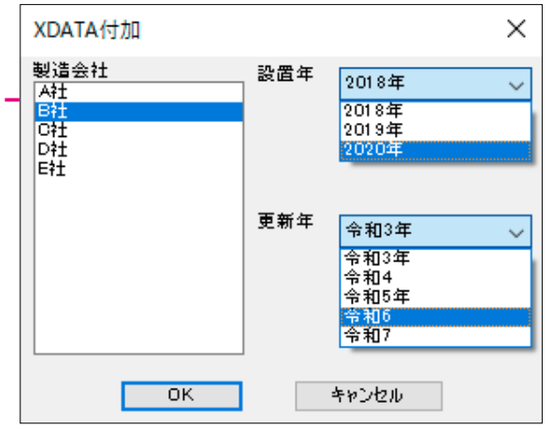


左図の○印の電球に拡張データ (Xdata) を付加する



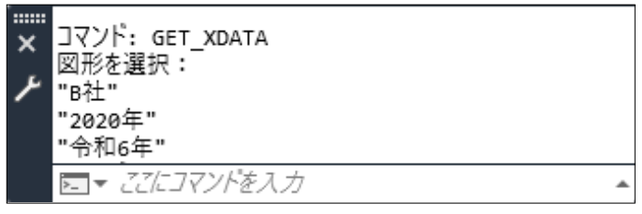
Step2 ーダイアログのリストから付加する文字を選択して、[Ok] ボタンを押します。
下図では、左上の電球を選択しています。
これだけで、ブロック以外の図形にもユーザー独自の情報を付加できます。





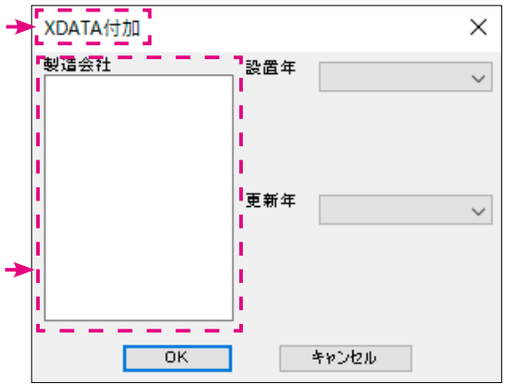
Step3 ー次の AutoLISP を使って、実際に拡張データ (Xdata) が付加されていることを確認します。

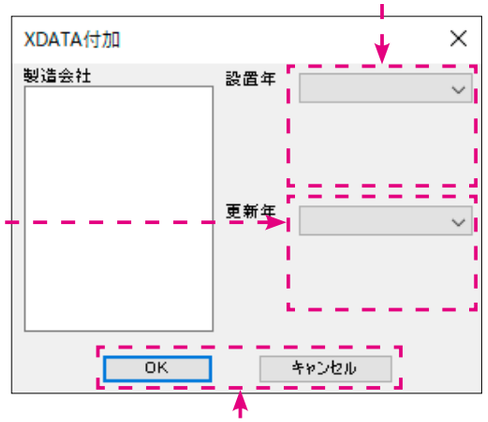
```
(defun c:get_xdata ( / ent ent1 en en1 en2 en3 xdata1 xdata2 xdata3)
  (setq ent (car (entsel "¥n 図形を選択："))) ; 拡張データが付加された図形を選択し、その図形情報を取得します。
  (if (/= ent nil)  ; 実際に選択された時（図形情報が存在するとき）、以下を実行します。
    (progn
; アプリケーション名 "MyApp" を含むデータを取得します。
      (setq ent1 (entget ent '("MyApp")))
;((-1 . < 図形名 : 7ef09b28>) (0 . "INSERT") (330 . < 図形名 : 7ef04cf8>) (5 . "6A5") (100 . "AcDbEntity") (67 . 0) (410 . "Model")
;(8 ." 電球 ") (62 . 7) (380 . 2) (100 . "AcDbBlockReference") (66 . 1) (2 . " 電球 ") (10 1179.85 1018.6 0.0) (41 . 1.0) (42 . 1.0) (43 . 1.0)
;(50 . 0.0) (70 . 0) (71 . 0) (44 . 0.0) (45 . 0.0) (210 0.0 0.0 1.0) (-3 ("MyApp" (1000 . "B 社 ") (1000 . "2020 年 ") (1000 . " 令和 6 年 ")))
      (setq en (cdr (assoc -3 ent1))); アプリケーション名が "MyApp" の Xdata だけを抽出します。
;(("MyApp" (1000 . "B 社 ") (1000 . "2020 年 ") (1000 . " 令和 6 年 ")) が取得できます。
        (if (/= en nil)
          (progn
            (setq enl (car en)) ;2 重の括弧 (( )) になっているので、外側の括弧 ( ) を外します。(car 関数で可能 )
;("MyApp" (1000 . "B 社 ") (1000 . "2020 年 ") (1000 . " 令和 6 年 ")) が取得できます。
            (setq en1 (cdr enl)) ;"MyAPP" の次から最後までを取得します。
;((1000 . "B 社 ") (1000 . "2020 年 ") (1000 . " 令和 6 年 ")) が取得できます。
            (setq xdata1 (cdr (assoc 1000 en1))) ; グループコードが <1000> のドット・ペア（1000 . "B 社 "）から
;2 番目の項目を取り出します。"B 社 " が取得できます。
            (setq en2 (cdr en1)) ;(1000 . "B 社 ") を省いた 2 番目以降を取り出します。
;((1000 . "2020 年 ") (1000 . " 令和 6 年 ")) が取得できます。
            (setq xdata2 (cdr (assoc 1000 en2))) ; グループコードが <1000> のドット・ペア（1000 . "2020 年 "）
; から 2 番目の項目を取り出します。"2020 年 " が取得できます。
            (setq en3 (cdr en2)) ;(1000 . "2020 年 ") を省いた 2 番目以降（残り）を取り出します。
;((1000 . " 令和 6 年 ")) が取得できます。
            (setq xdata3 (cdr (assoc 1000 en3))) ; グループコードが <1000> のドット・ペア（1000 . " 令和 6 年 "）
; から 2 番目の項目を取り出します。" 令和 6 年 " が取得できます。
            (print xdata1) ;xdata1<"B 社 "> をコマンドラインに改行して表示します。
            (print xdata2) ;xdata2<"2020 年 "> をコマンドラインに改行して表示します。
            (print xdata3) ;xdata3<" 令和 6 年 "> をコマンドラインに改行して表示します。
          );prog
        );if
      );prog
    );if
(princ)
);end
```



使用するダイアログ	S_Popup2.dcl

```
S_Popup2 :dialog
{
  label = "XDATA 付加 ";
  :row              /* タイルを横並びに配置スタート */
  {
    :list_box   /* リストタイル配置スタート */
    {
      label = " 製造会社 ";
      key = "list1";
      width = 20;
      height = 15;
    }           /* リストタイル配置終了 */
   :column   /* タイルを縦並びに配置スタート */
   {
      :popup_list   /* Popup リストタイル配置スタート */
      {
        label = " 設置年 ";
        key = "list2";
        width = 15;
        value = 0;
      }         /* Popup リストタイル配置終了 */
      :popup_list   /* Popup リストタイル配置スタート */
      {
        label = " 更新年 ";
        key = "list3";
        width = 15;
        value = 0;
      }         /* Popup リストタイル配置終了 */
    spacer;
    }           /* タイルを縦並びに配置終了 */
  }             /* タイルを横並びに配置終了 */

  /* OK or CANCEL */
  spacer;
  ok_cancel;
}//end
```





使用する LISP	S_Popup2.lsp

```
(defun C:S_Popup2( / list1_data list2_data list3_data sel_list1 sel_list2 sel_list3)
;<<List1 のリストを作成 >>
❶ (setq list1_data (list "A 社 " "B 社 " "C 社 " "D 社 " "E 社 "))
List2 のリストを作成
(setq list2_data (list "2018 年 " "2019 年 " "2020 年 "))
List3 のリストを作成
(setq list3_data (list " 令和 3 年 " " 令和 4 年 " " 令和 5 年 "
                       " 令和 6 年 " " 令和 7 年 "))
```

各リストに流し込むデータを予めリスト化しておく

```
;<< 各 List の選択した番号をセット ( 一番上が <"0"> になります）>>
Point!
❷ (defun SaveTileVal()
   (setq sel_list1(nth (atoi(get_tile "list1")) list1_data)) ; 選択した項目の番号
   (setq sel_list2(nth (atoi(get_tile "list2")) list2_data)) ; 選択した項目の番号
   (setq sel_list3(nth (atoi(get_tile "list3")) list3_data)) ; 選択した項目の番号
   )
```

リストの中の選択した番号を取得し、整数に変換する

```
;<< ダイアログを使用する準備 >>
❸ (setq dcl_id (load_dialog "S_Popup2.dcl")) ;DCL をメモリーに読み込む
   (new_dialog "S_Popup2" dcl_id)            ;S_Popup2 ダイアログを初期化する
```

ダイアログの読み込みと表示

```
;<<list1 に list1_data を流し込む >>
❹ (start_list "list1" 3)                    ; リスト "list1" を新規でスタート
   (mapcar 'add_list list1_data)            ; リスト "list1" 内に <lst1_data> の内容を流し込む
   (end_list)                               ; "list1" へのリスト追加を終了
```

"list1" 内に、予め作成した <list1_data> を流し込む

```
;<<list2 に list2_data を流し込む >>
❺ (start_list "list2" 3)                    ; リスト "list2" を新規でスタート
   (mapcar 'add_list list2_data)            ; リスト "list2" 内に <lst2_data> の内容を流し込む
   (end_list)                               ; "list2" へのリスト追加を終了
```

"list2" 内に、予め作成した <list2_data> を流し込む

```
;<<list3 に list3_data を流し込む >>
❻ (start_list "list3" 3)                    ; リスト "list3" を新規でスタート
   (mapcar 'add_list list3_data)            ; リスト "list3" 内に <lst3_data> の内容を流し込む
   (end_list)                               ; "list3" へのリスト追加を終了
```

"list3" 内に、予め作成した <list3_data> を流し込む

Point!

❷ (nth (atoi(get_tile "list1")) list1_data)

"list1" 内のリストの項目が選択された時は、その項目の番号が文字で取得されます。
例えば、上から 2 番目が選択された時は、<"3"> がセットされます。
（番号は <"0"> 番から始まります。）
nth 関数は、リストの何番目（整数）の項目を取得するか・・ですから、この文字 <"3"> を整数に変換する必要があります。(atoi "3")
その番号に符合するリスト内の文字を取得して、変数 [sel_list1] にセットしています。


```
❼ (action_tile "accept"  "(SaveTileVal)(done_dialog 1)") ; 内部サブ関数 (SaveTileVal) を実行して終了
   (action_tile "cancel"  "(done_dialog 0)")              ; 何も処理しないで閉じる
;<< ダイアログを終了する準備 >>
❽ (setq chkdia (start_dialog))  ; ユーザー入力を受け付ける
   (unload_dialog dcl_id)       ; ダイアログをメモリーから解放する
```

ダイアログの入力と終了

```
;chkdia には、done_dialog の status( 値 ) が入っています。[Ok] は <1>、[Cancel] は <0> です。
❾ (if(= chkdia 1)(S_Popup2_1)) ;[Ok] ボタンが押された時の処理

 (princ)
);end

;<< 外部サブ関数 >>
❿ (defun S_Popup2_1()
 ;<< 拡張エンティティとして情報を付加する >>
 (setq ent1 (car (entsel "¥n 図形を選択："))) ; 図形を選択して、図形名を取得する
 (if (/= ent1 nil) ; もし図形が選択されていれば、以下を実行する
  (progn
   (regapp "MyApp") ; 使用するアプリケーション名を付加する
;sel_list1、sel_list2、sel_list3 の文字をそれぞれ 1000 でドット・ペアを作成する
Point!
   (setq old_xdata (list (list -3 (list "MyApp" (cons 1000 sel_list1)
                   (cons 1000 sel_list2) (cons 1000 sel_list3)))))
   (setq ent2 (entget ent1)) ; 図形名 ent1 のリスト形式の定義データを取得する
   (setq new_xdata (append ent2 old_xdata)) ;3 つの Xdata を追加する
   (entmod new_xdata) ; 図形情報を再定義する
  );progn
 );if
 (princ)
);end
```

Point!

❿ (list (list -3 (list "MyApp" (cons 1000 sel_list1))))

拡張データ (Xdata) は 1 つ以上のグループから構成されていて、アプリケーション名で始まります。（本プログラムでは、"MyApp"）

この拡張データの始まりは、コード <-3> で示されます。<-3> に続いて、アプリケーション名と付加するデータを list 関数で選択した図形の定義リストに連結します。

付加するデータが文字の場合のドット・ペアは <1000> です。
ドット・ペアを作成する cons 関数を使って、(cons 1000 " 会社 A") のように記述します。
" 会社 A" が変数になっている場合は、(cons 1000 sel_list1) になります。

番号	説　明
①	"list1" から "list3" に使用するリストを事前に作成します。
②	リスト内で選択した番号を整数に変換して取得します。番号はリストの上から <0> から順にカウントします。
③	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
④	"list1" 内に [list1_data] を流し込みます。mapcar は [list1_data] をリスト内に順番にリストとして追加していく関数です。[start_list] は [load_dialog] の後、[action_tile] の前に記述します。
⑤	④と同様に、[list2_data] を "list2" 内にリストとして流し込みます。[start_list] は [load_dialog] の後、[action_tile] の前に記述します。
⑥	④と同様に、[list3_data] を "list3" 内にリストとして流し込みます。[start_list] は [load_dialog] の後、[action_tile] の前に記述します。
⑦	[OK] ボタンが押されると、内部サブ関数 (SaveTileVal) を起動して、フラッグ (status) として <1> を指定します。 この数値はユーザーが指定できます。
⑧	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑨	status が <1> であれば、外部サブ関数 (S_Popup2_1) を起動します。<0> であれば何もせずに終了します。
⑩	外部サブ関数 (S_Popup2_1) を起動します。この時点で、前のダイアログに戻ることはできません。

Point!

Xdata を付加した直後であれば、キーボードから [!new_xdata] と入力すると
下図のようにコマンドラインに情報が表示されます。
最後の (-3) 以下に拡張データ (Xdata) の内容が付加されていることが判ります。

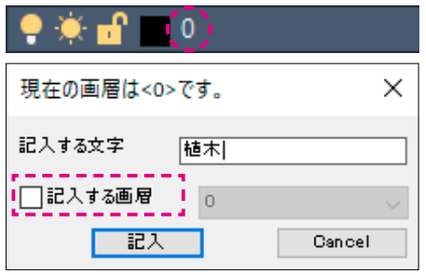
```
コマンド: !new_xdata
((-1 . <図形名: 1f7f18d6850>) (0 . "HATCH") (330 . <図形名: 1f7abb0b9f0>)
(5 . "6E5") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "電球")
(62 . 0) (380 . 2) (100 . "AcDbHatch") (10 0.0 0.0 0.0) (210 0.0 0.0 1.0)
(2 . "ANSI31") (70 . 0) (71 . 0) (91 . 1) (92 . 7) (72 . 1) (73 . 1) (93 .
6) (10 1013.73 1325.26 0.0) (42 . 0.0) (10 1013.73 1321.54 0.0) (42 . 0.0)
(10 1016.71 1321.54 0.0) (42 . 0.0) (10 1016.71 1324.76 0.0) (42 .
-0.00507975) (10 1019.94 1332.94 0.0) (42 . 1.52719) (10 1010.26 1332.69
0.0) (42 . 0.0) (97 . 0) (75 . 0) (76 . 1) (52 . 0.0) (41 . 1.0) (77 . 0)
(78 . 1) (53 . 0.785398) (43 . 1062.96) (44 . 1008.74) (45 . -0.0883883)
(46 . 0.0883883) (79 . 0) (47 . 0.991871) (98 . 1) (10 1014.97 1333.93 0.0)
(-3 ("MyApp" (1000 . "B社") (1000 . "2020年") (1000 . "令和6年"))))
ここにコマンドを入力
```


第 11 節　指定した画層に文字を記入する

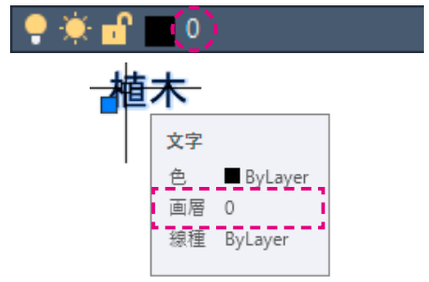
使用する LISP	S_Toggle2.lsp	使用する主な関数	tblnext、append、nth、mapcar
内部サブ関数	(SaveTileVal)	外部サブ関数	なし
使用する DCL	S_Toggle2.dcl	使用するタイル	text、edit_box、toggle、popup_list
外部テキスト	なし	スライドファイル	なし
このプログラムは、図面の中に挿入する文字の画層を自由に選択できるようにしたものです。 挿入する画層を切り替えることなく、文字を挿入できます。			

Step1 ー <S_Toggle2.lsp> を起動すると、下図のダイアログが表示されます。
ダイアログのトップに現在の画層名が表示されます。
このままの画層で良ければ、文字を書き入れた後、[記入] ボタンを押して文字を挿入します。
文字の位置はユーザーが指定します。

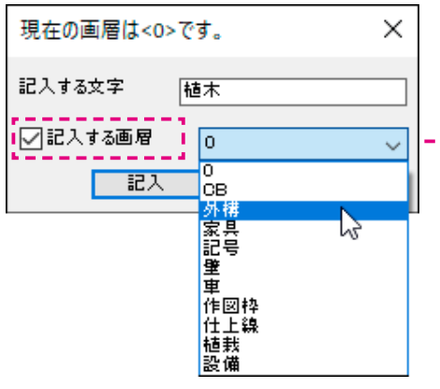
[記入する画層] をオフにして文字を挿入

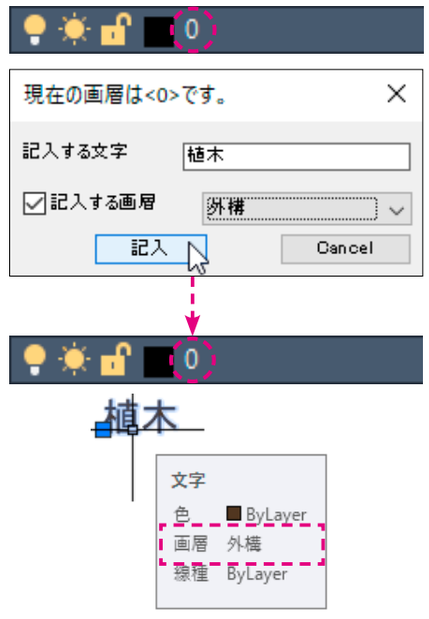


現在画層 <0> に文字が挿入されました



Step2 ー [記入する画層] をオンにすると、リストに画層一覧が表示されます。(左図)
この例では、挿入画層を < 外構 >、挿入文字を < 植木 > としています。(右図)
画層 [外構] に文字が挿入されましたが、現在画層は [0] のままです。(右下図)

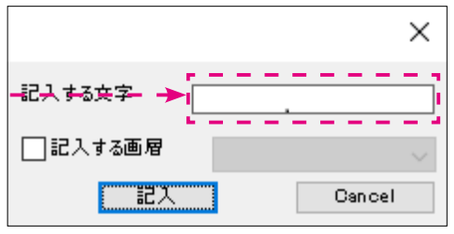


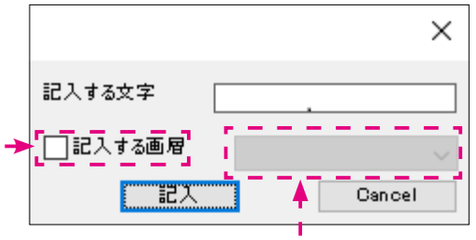


使用するダイアログ	S_Toggle2.dcl

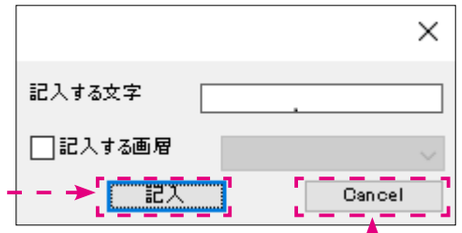
```
S_Toggle2 :dialog
{
 key = "Title";
 label = "";
 :column    /* タイルを縦並びに配置スタート */
 {
 spacer;
  :row       /* タイルを横並びに配置スタート */
  {
   :text
   {
    label = " 記入する文字 ";
   }
   :edit_box  /* 編集ボックス処理タイルスタート */
   {
    key = "text1";
    width = 18;
    value = "";
   }        /* 編集ボックス処理タイル終了 */
  }         /* タイルを横並びに配置終了 */

 spacer;
 :row      /* タイルを横並びに配置スタート */
 {
   :toggle
   {
    key = chk1;
    label = " 記入する画層 ";
    value = "0";
   }
   :popup_list
   {
    key = layer_sel;
    width = 20;
   }
 }   /* タイルを横並びに配置終了 */
}   /* タイルを縦並びに配置終了 */
```

Point!（ラベルは LISP で記述）






```
/* OK or CANCEL */
:row    /* タイルを横並びに配置スタート */
{
  spacer;
  : button
  {
     label = " 記入 ";
     is_default = true;
     key = "accept";
     width = 8;
     fixed_width = true;
  }

  : button
  {
     label = "Cancel";
     is_cancel = true;
     key = "Cancel";
     width = 4;
     fixed_width = true;
  }
   }  /* タイルを横並びに配置終了 */
}//end
```

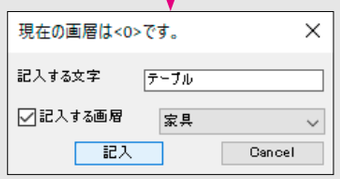


Point!

タイルのラベルはダイアログ側か AutoLISP 側で指定できます。

ダイアログで指定する

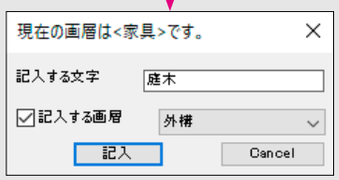
```
key = "Title";
 label = " 現在の画層は <0> です。";
```



タイトルが固定されてしまうので、実際の画層名と異なる場合もあります。

AutoLISP で指定する

```
(setq c_layer (getvar "Clayer"))
(setq c_layer1 (strcat " 現在の画層は <" c_layer "> です。"))
(set_tile "Title" c_layer1)
```



状況に応じて、適切なラベルを表現できます。ラベルを固定化出来ない場合に有効です。

使用する LISP	S_Toggle2.lsp

```
(defun C:S_Toggle2( / layname laydata lst sel_layer dcl_id c_layer
                      c_layer1 text1 chkdia pt1 takasa)

;<< 画層のリストを作成 >>
❶ (setq layname (tblnext "LAYER" T))          ; 最初の画層を取得する
  (while layname                              ; layname が nil を返すまで繰り返す
  (setq laydata (cdr (assoc 2 layname)))      ; layname の画層名を取得する
  (setq lst (append lst (list laydata)))      ; 取得した画層名をリストにする
  (setq layname (tblnext "LAYER")))           ; 次の画層を取得 ( 無ければ nil を返す )
                                              → 図面内にある画層のリストを作成する
;<< ダイアログを使用する準備 >>
❷ (setq dcl_id (load_dialog "S_Toggle2.dcl"))  ;S_Toggle2.dcl をメモリーに読み込む
  (new_dialog "S_Toggle2" dcl_id)              ;S_Toggle2 ダイアログを初期化する
                                              → ダイアログの読み込みと表示
Point!
;<< 初期値として、画層名を非表示にしておく（チェックボックスで ON/OFF を切り替えるため）>>
❸ (mode_tile "layer_sel" 1)                   ;"layer_sel"（画層の popup）を選択できない状態にしておく

;<< リスト内に画層の一覧を流し込む >>
❹ (start_list "layer_sel" 3)                  ; リスト "layer_sel" を新規でスタート
  (mapcar 'add_list lst)                      ; リスト "layer_sel" 内に <lst> の内容を流し込む
  (end_list)                                  ; "layer_sel" へのリスト追加を終了
                                              → "layer_sel" 内に、予め作成した <list> を流し込む
Point!
;<<toggle の ON/OFF をチェックする >>
❺ (action_tile "chk1" "(mode_tile ¥"layer_sel¥" (- 1 (atoi (get_tile ¥"chk1¥"))))")

;<< リスト内の選択された画層名を調べる >>
❻ (defun SaveTileVal()
    (setq sel_layer(nth (atoi(get_tile "layer_sel")) lst))  ; 選択されたリスト番号から画層名を取得
  )

;<< タイトルに現在の画層名を表示する >>
❼ (setq c_layer (getvar "Clayer"))            ; システム変数から現在画層名を取得
  (setq c_layer1 (strcat " 現在の画層は <" c_layer "> です。"))  ; 文字列を連結する
  (set_tile "Title" c_layer1)                 ;"Title" に連結した文字列を表示する
```

Point!

❸ [記入する画層] の初期値は、チェックされていない <"0"> の状態なので、popup の "layer_sel" も非表示 <1> にしておきます。

mode_tile の mode のコード	
値	説　明
0	タイルを使用可能にします。
1	タイルを使用禁止にします。
2	タイルにフォーカスを設定します。
3	編集ボックスの内容を選択します。
4	イメージのハイライト表示のオンとオフを切り替えます。


```
❽ (action_tile "text1" "(setq text1 (get_tile ¥"text1¥"))")  ;"text1" 内の文字を取得する
;[OK] ボタンと [Cancel] ボタンの動作を記述する (done_dialog) の status をセット
❾ (action_tile "accept"  "(SaveTileVal)(done_dialog 1)")  ; 内部サブ関数 (SaveTileVal) を実行して終了
  (action_tile "cancel"  "(done_dialog 0)")               ;status を <0> にセットして終了
;<< ダイアログを終了する準備 >>
❿ (setq chkdia (start_dialog))                ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                      ; ダイアログをメモリーから解放する
                                              → ダイアログの入力と終了

;<<[OK] ボタンが押された時の分岐処理 >>
⓫ (if(= chkdia 1)                            ;[ 記入 ] ボタンが押された時の処理を以下に記述する
   (if sel_layer   ; 画層名が指定されたときの処理
    (progn
     (command "-layer" "s" sel_layer "")   ; 選択した画層に変更する
     (setq pt1 (getpoint "¥n 挿入位置を指示："))   ; 文字の挿入位置を指定させる
     (setq takasa (getreal "¥n 文字の大きさ："))   ; 文字の大きさを指定させる
     (command "text" pt1 takasa "" text1)   ;[ ダイナミック文字記入 ] で選択した文字を挿入する
     (command "-layer" "s" c_layer "")   ; 画層を現在層に戻す
    );progn
    (progn   ; 画層名が指定されないときの処理
     (setq pt1 (getpoint "¥n 挿入位置を指示："))   ; 文字の挿入位置を指定させる
     (setq takasa (getreal "¥n 文字の大きさ："))   ; 文字の大きさを指定させる
     (command "text" pt1 takasa "" text1)   ;[ ダイナミック文字記入 ] で選択した文字を挿入
    );progn
   );if
  );if
  (princ)
);end
```

Point!

❺ (action_tile "chk1" "(mode_tile ¥"layer_sel¥" (- 1 (atoi (get_tile ¥"chk1¥"))))")

① (get_tile ¥"chk1¥")・・"chk1" の値は、"0" か "1" です。
② (atoi (get_tile ¥"chk1¥"))・・減算処理を行うために、整数に変換します。
③ (- 1 (atoi (get_tile ¥"chk1¥")))・・もし、"chk1" が "0" の時は、この中の値は <1> になります。
　もし、"chk1" が "1" の時は、この中の値は <0> になります。
④ [記入する画層] がチェックされるたびに、"layer_sel" の mode は反転し、<On> と <Off> が切り替わります。

(注）最初に [記入する画層] の "chk1" と "layer_sel" は両方とも <Off> の状態にしておかないと、逆の結果になってしまいます。

番号	説　明
①	図面内にある画層名を収集します。 tblnext 関数で 1 番目の画層名を取得します。それが存在した時に <while> 以下を実行します。 （tblnext 関数の使い方は、P2-143<Point!> を参考） (cdr (assoc 2 layname)) で得た画層名を変数 <laydata> にセットします。 append 関数で、こうして得た画層名を追加してリストを作成します。
②	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
③	画層を表示する "layer_sel" の表示を開始時は非表示にしておきます。toggle_button と同期性を保つためです。
④	リスト内容の表示は、[start_list][add_list][end_list] の 3 つを記述します。 mapcar 関数で画層名を順番にリスト内に流し込んでいきます。
⑤	toggle_button の "chk1" が押される度に、"layer_sel" の表示 / 非表示を切り替えます。
⑥	選択されたリスト番号から、その番号に対応する画層名を取得します。
⑦	現在の画層名を取得し (getvar "Clayer")、その名をタイトルのラベル <Title> に代入します。
⑧	"text1" に記入された文字列を取得します。[記入] ボタンが押された時に、挿入される文字になります。
⑨	[OK] ボタンが押されると、内部サブ関数 (SaveTileVal) を起動して、フラッグ (status) として <1> を指定します。 この数値はユーザーが指定できます。
⑩	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑪	フラッグ (status) が <1> であれば、if 以下を実行します。<0> であれば何もせずに終了します。

Point!

❺ タイルの <On> と <Off> を反転させるには、下記の①と②の方法があります。

① (action_tile "chk1" "(mode_tile ¥"layer_sel¥" (- 1 (atoi (get_tile ¥"chk1¥"))))")
本プログラムで記述した方法です。減算処理により、popup_tile "layer_sel" の mode を反転しています。

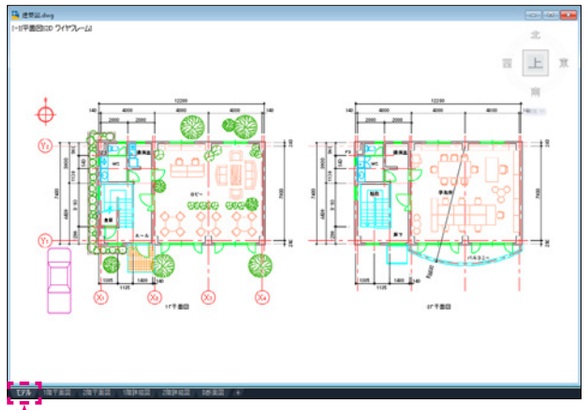
② toggle_button "chk1" が <On> か <Off> された時の処理をサブ関数に記述する方法です。

```
;toggle "chk1" が変更された時、サブ関数 (chk_toggle) を起動する
(action_tile "chk1" "(chk_toggle)")
; サブ関数を以下に記述
(defun chk_toggle()
;toggle_button "chk1" の値 <"0"> か <"1"> を整数に変換して変数 [chk] にセット
   (setq chk (atoi (get_tile "chk1")))
     (if (= chk 1)                    ; もし、[chk] の値が <1> の時は、以下の処理を行う。
       (mode_tile "layer_sel" 0)    ;popup_tile "layer_sel" の mode を <0>（表示）にする
       (mode_tile "layer_sel" 1)   ;<1> でなければ、"layer_sel" の mode を <1>（非表示）にする
      );if
  );end
```

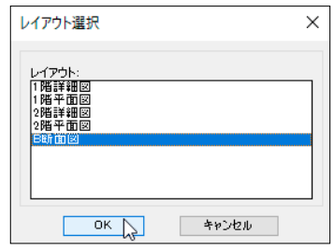

第 12 節　レイアウトページを選択する

使用する LISP	S_List1.lsp	使用する主な関数	nth、acad_strsort、mapcar
内部サブ関数	(SaveTileVal)	外部サブ関数	なし
使用する DCL	S_List1.dcl	使用するタイル	list_box
外部テキスト	なし	スライドファイル	なし
このプログラムは、図面のレイアウトをダイアログから変更できるようにしたものです。 レイアウトの数が多い図面には便利です。			

Step1 ー <S_List1.lsp> を起動すると、右下図のダイアログが表示されます。
ダイアログには、図面にあるレイアウト名の一覧が表示されます。
選択するレイアウト名を選んで、[OK] ボタンを押します。

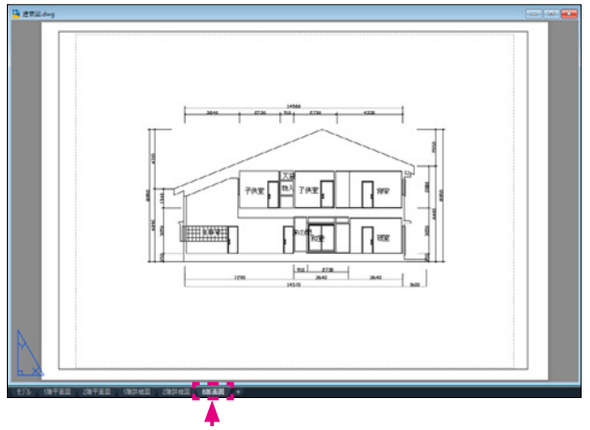


現在は [モデル空間] です。

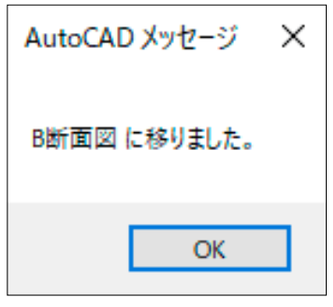


レイアウト [B 断面図] を選択します。

Step2 ー選択したレイアウトに移り、確認のためのレイアウト名が表示されます。



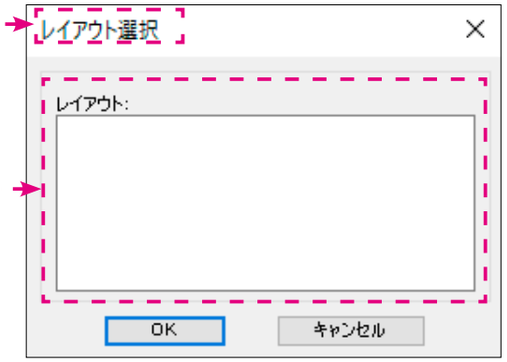
[B 断面図] のレイアウトに移りました。

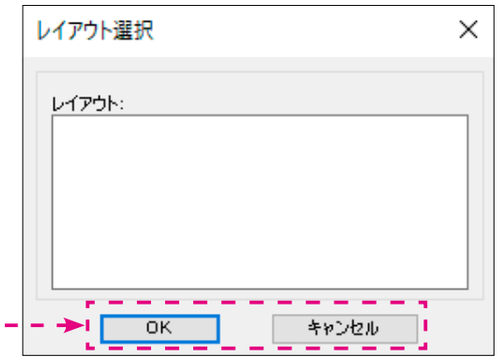


使用するダイアログ	S_List1.dcl

```
S_List1 : dialog
{
 label = " レイアウト選択 ";
 : column    /* タイルを縦並びに配置スタート */
 {
  : boxed_column  /* タイルを縦並びに配置スタート */
  {
   : list_box
   {
    label = " レイアウト :";
    key = "lay";
    multiple_select = false;
    width = 40;
   }
  }   /* タイルを縦並びに配置終了 */
 }   /* タイルを縦並びに配置終了 */
  /* OK or CANCEL */
        ok_cancel;
}//end
```

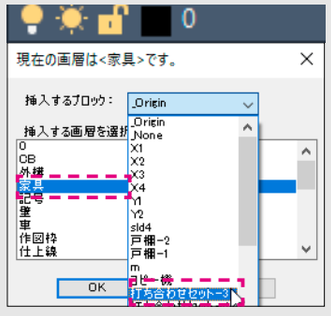
Point!





Point!

List_box の [multiple_select = false;] と popup_list



list_box の [multiple_select = false;] と popup_list は
どちらも選択が 1 つしか出来ないことが共通です。

選択肢を 1 つに限るのは、[画層の切り替え]、[ビューの切り替え] 等があります。
もし、複数の項目の選択を可能にしたい（ブロック挿入や外部からの画層名の取り込み）時は、[multiple_select = true;] と記述します。

使用する LISP	S_List1.lsp

```
(defun C:S_List1( / sel_layout)

;<< レイアウトがあるかどうかをチェックする。>>
(if (setq layname(layoutlist))                ; レイアウトが有れば、以下を実行する
 (progn
  (setq layname(acad_strlsort layname))       ; レイアウト名を昇順で並び替える
  (setq dcl_id (load_dialog "S_List1.dcl"))   ;S_List1.dcl をメモリーに読み込む
  (new_dialog "S_List1" dcl_id)               ;S_List1 ダイアログを初期化する
 )
)
;<< リスト内にレイアウト名を流し込む >>
 (start_list "lay" 3)                         ; リスト "lay" を新規でスタート
 (mapcar 'add_list layname)                   ; リスト "lay" 内に <layname> の内容を流し込む
 (end_list)                                   ; リスト "lay" へのリスト追加を終了

 (defun SaveTileVal()
  (setq sel_layout(nth (atoi(get_tile "lay")) layname))  ; 選択されたリスト番号から
 )                                                       ; レイアウト名を取得
;[OK] ボタンと [Cancel] ボタンの動作を記述する (done_dialog) の status をセット
 (action_tile "accept"  "(SaveTileVal)(done_dialog 1)")  ; 内部サブ関数 (SaveTileVal) を実行する
 (action_tile "cancel"  "(done_dialog 0)")               ;status を <0> にセットして終了
;<< ダイアログを終了する準備 >>
 (setq chkdia (start_dialog))                ; ユーザー入力を受け付ける
 (unload_dialog dcl_id)                      ; ダイアログをメモリーから解放する
```

❶ レイアウトが有れば、ダイアログに表示する

❷ "lay" 内に、作成したリストを流し込む

❸ ❹ ダイアログの入力と終了

Point!

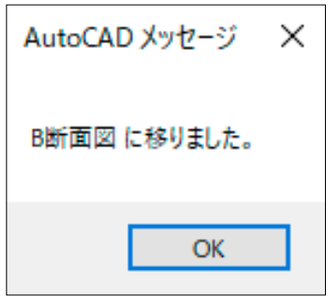
❶ **(setq layname(layoutlist))**

→ (layoutlist) は AutoCAD の組み込み関数です。レイアウト名を収集します。
コマンドラインから入力する場合は、(layoutlist) と ( ) 付きで入力します。
(例）変数 layname には、("1 階詳細図 " "2 階詳細図 ") のような形でセットされます。

acad_strlsort

→ (acad_strlsort< 文字列のリスト >) は AutoCAD の組み込み関数です。
< 文字列のリスト > を昇順で並び替えます。
（例）変数 layname が、("2 階詳細図 " "1 階詳細図 ") であれば、
("1 階詳細図 " "2 階詳細図 ") に並び替えられます。
（例）(acad_strlsort (list "3" "2" "1")))　→ ("1" "2" "3")

```
;<<[OK] ボタンが押された時の分岐 >>
❻ (if(= chkdia 1) ;[OK] ボタンが押された時の処理
    (if sel_layout  ;もしレイアウト名が選択されていれば、以下を実行する
      (progn
Point!  (command "layout" "s" sel_layout)  ;そのレイアウトに移動する
        (alert (strcat sel_layout " に移りました。"))  ;メッセージを表示する
      );progn
    );if
  );if
 );progn
);if

(princ)
);end
```



Point!

❻ command "layout" "s" sel_layout)

キーボードから <layout> と入力すると、以下のメッセージが表示されます。

コマンド : layout
レイアウトのオプションを入力 [複写 (C)/ 削除 (D)/ 新規作成 (N)/ テンプレート (T)/ 名前変更 (R)/ 名前を付けて保存 (SA)/ 設定 (S)/ 一覧 (?)] < 設定 >:

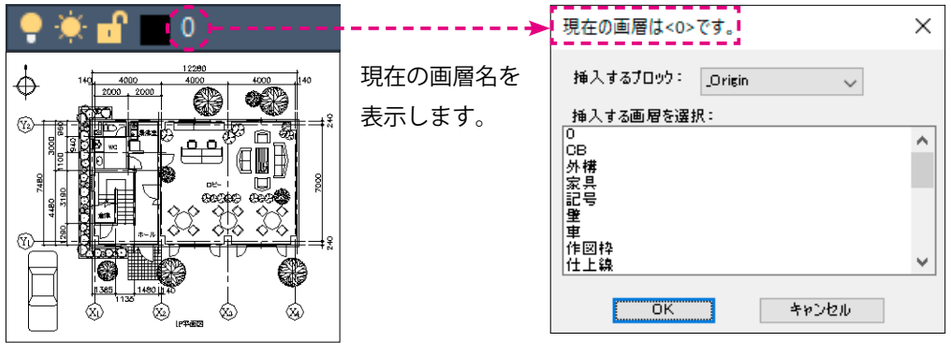
この中のオプション [設定 (S)] を使って、レイアウトの切り替えが出来ます。

番号	説　明
①	現在の図面内にレイアウトが有れば、dcl ファイルを読み込みます。(layoutlist) はレイアウト名を取得する関数です。 dcl ファイルには複数のダイアログを含めることができます。dcl ファイルの中のダイアログを初期化します。 指定したダイアログが無い場合、(exit) の処理も記述できます。
②	①で取得したレイアウト名のリストを "lay" 内に流し込みます。
③	リスト内で選択した番号を整数に変換して取得します。番号はリストの上から順に <0> からカウントします。 その番号に相応したレイアウト名を変数 <sel_layout> にセットします。
④	[OK] ボタンが押されると、内部サブ関数 (SaveTileVal) を起動して、フラッグ (status) として <1> を指定します。 この数値はユーザーが指定できます。
⑤	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑥	status が <1> であれば、if 以下を実行します。<0> であれば何もせずに終了します。

第 13 節　指定した画層にブロックを挿入する

使用する LISP	S_Popup1.lsp	使用する主な関数	tblnext、append、nth、mapcar
内部サブ関数	(SaveTileVal)	外部サブ関数	なし
使用する DCL	S_Popup1.dcl	使用するタイル	text、popup_list、list_box
外部テキスト	なし	スライドファイル	なし
このプログラムは、図面の中に挿入するブロックの画層を自由に選択できるようにしたものです。 挿入する画層に切り替えることなく、ブロックを挿入できます。			

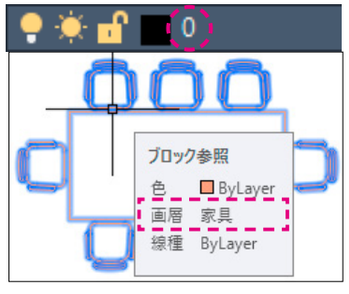
Step1 ー <S_Popup1.lsp> を起動すると、右下図のダイアログが表示されます。
起動した時点で、ダイアログのタイトルには現在画層が表示されます。
また、リスト内に図面の画層一覧とポップアップにブロックの一覧が表示されます。



現在の画層名を表示します。

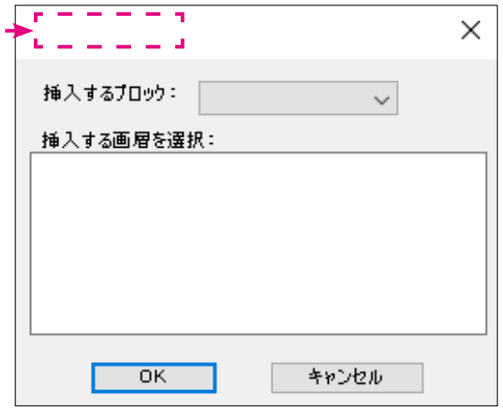
Step2 ー現在の画層は <0> です。挿入するブロックを < 打ち合わせセット > を選択します。
挿入する画層に < 家具 > を選びます。

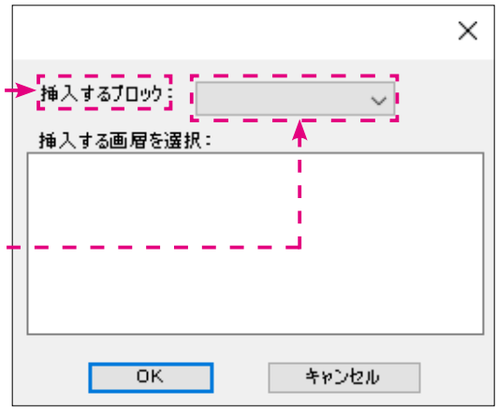
Step3 ーブロックは画層 < 家具 > に挿入されましたが、現在画層は <0> のままです。

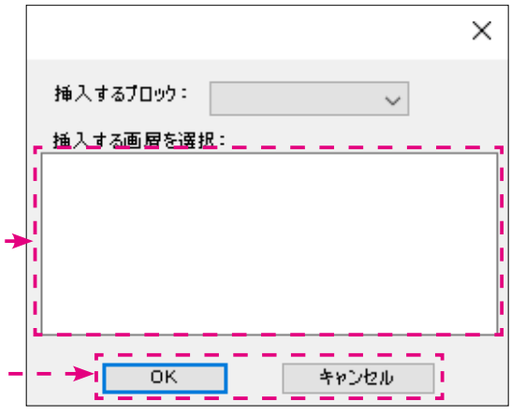


使用するダイアログ	S_Popup1.dcl

```
S_Popup1 : dialog
{
 key = "title";
 label = "";
 spacer;
 : column    /* タイルを縦並びに配置スタート */
 {
  fixed_width = true;
  : row    /* タイルを横並びに配置スタート */
  {
   : text
   {
    label = " 挿入するブロック：";
    key = "text1";
    width = 10;
    fixed_width = true;
    spacer;
   }
   : popup_list
   {
    key = "block_sel";
    width = 15;
    fixed_width = true;
   }
  }    /* タイルを横並びに配置終了 */
 spacer;
 : list_box
 {
  label = "    挿入する画層を選択：";
  key = "layer_sel";
  multiple_select = false;
  width = 42;
  fixed_width = true;
 }
}    /* タイルを縦並びに配置終了 */
 spacer;
 ok_cancel;
}//end
```







使用する LISP	S_Popup1.lsp

```
(defun C:S_Popup1( / blkname blkdata blklst layname laydata laylst  sel_block
                     sellayer dcl_id c_layer  c_layer1 chkdia pt1)
;<< ブロック図形のリストを作成 >>
(setq blkname (tblnext "BLOCK" T))                    ; 最初のブロック名を取得する
(while blkname                                        ; blkname が nil を返すまで繰り返す
  (setq blkdata (cdr (assoc 2 blkname)))              ; blkname のブロック名を取得する
  (setq blklst (append blklst (list blkdata)))        ; 取得したブロック名をリストにする
  (setq blkname (tblnext "BLOCK"))                    ; 次のブロックを取得 ( 無ければ nil を返す )
)                                                     → 図面内にあるブロックのリストを作成する
;<< 画層のリストを作成 >>
(setq layname (tblnext "LAYER" T))                    ; 最初の画層名を取得する
(while layname                                        ; layname が nil を返すまで繰り返す
  (setq laydata (cdr (assoc 2 layname)))              ; layname の画層名を取得する
  (setq laylst (append laylst (list laydata)))        ; 取得した画層名をリストにする
  (setq layname (tblnext "LAYER"))                    ; 次の画層を取得する ( 無ければ nil を返す )
)                                                     → 図面内にある画層のリストを作成する
;<< リスト内の選択されたブロック名と画層名を調べる >>
(defun SaveTileVal()
  (setq sel_block(nth (atoi(get_tile "block_sel")) blklst))  ; リスト番号からブロック名を取得
  (setq sel_layer(nth (atoi(get_tile "layer_sel")) laylst))  ; リスト番号から画層名を取得
)
;<< ダイアログを使用する準備 >>
(setq dcl_id (load_dialog "S_Popup1.dcl"))            ;S_Popup1.dcl をメモリーに読み込む
(new_dialog "S_Popup1" dcl_id)                        ;S_Popup1 ダイアログを表示する
                                                      → ダイアログの読み込みと表示
;<< タイトルを表示する（現在の画層）>>
(setq c_layer (getvar "Clayer"))
(setq c_layer1 (strcat " 現在の画層は <" c_layer "> です。"))
(set_tile "title" c_layer1)                           → 現在の画層名をダイアログのラベルに記述
```

Point!

❶ (setq blkname (tblnext "BLOCK" T))

最初に取得したブロックが以下であるとすると、（変数名は blkname）

((0 . "BLOCK") (2 . "_Origin") (70 . 0) (4 . "") (10 0.0 0.0 0.0) (-2 . < 図形名 : 7ef04e38>))

これからブロック名を取得します。

(setq blkdata (cdr (assoc 2 blkname)))　→ "_Origin"

まず、最初のブロック名をリストにします。

(setq blklst (append blklst (list blkdata))　→ ("_Origin")

次のブロック名が存在すれば、リストへの追加を繰り返します。

```
;<< リスト内にブロックの一覧を流し込む >>
(start_list "block_sel" 3)   ; リスト "block_sel" を新規でスタート
(mapcar 'add_list blklst)    ; リスト "block_sel" 内に <blklst> の内容を流し込む
(end_list)                   ; "block_sel" へのリスト追加を終了
```

⑥ "block_sel" 内に、予め作成した <blklst> を流し込む

```
;<< リスト内に画層の一覧を流し込む >>
(start_list "layer_sel" 3)   ; リスト "layer_sel" を新規でスタート
(mapcar 'add_list laylst)    ; リスト "layer_sel" 内に <laylst> の内容を流し込む
(end_list)                   ; "layer_sel" へのリスト追加を終了
```

⑦ "layer_sel" 内に、予め作成した <laylst> を流し込む

```
;[OK] ボタンと [Cancel] ボタンの動作を記述する (done_dialog) の status をセット
(action_tile "accept"  "(SaveTileVal)(done_dialog 1)")  ; 内部サブ関数 (SaveTileVal) を実行する
(action_tile "cancel"  "(done_dialog 0)")               ;status を <0> にセットして終了
```

⑧

```
;<< ダイアログを終了する準備 >>
(setq chkdia (start_dialog))  ; ユーザー入力を受け付ける
(unload_dialog dcl_id)        ; ダイアログをメモリーから解放する
```

⑨ ダイアログの入力と終了

```
;<<[OK] ボタンが押された時の分岐 >>
(if(= chkdia 1)  ;[OK] ボタンが押された時の処理を以下に記述する
    (if sel_block   ; ブロックが選択された時の処理
        (progn
         (command "-layer" "s" sel_layer "")   ; 選択した画層に変更する
         (setq pt1 (getpoint "¥n 挿入位置を指示："))   ; ブロックの挿入位置を指定させる
         (command "-insert" sel_block pt1 "" "" "")   ; ブロックを挿入する
         (command "-layer" "s" c_layer "")   ; 画層を現在層に戻す
        );progn
    );if
  );if
 (princ)
);end
```

⑩

Point!

(command "-insert" sel_block pt1 "" "" "")

[- insert] と入力すると、コマンドラインには以下のように表示されます。

- ブロック名を入力 または [一覧 (?)]:　→ 変数 [sel_block] の値をセットします。
- 挿入位置を指定 または [基点 (B)/ 尺度 (S)/X/Y/Z/ 回転 (R)]:　→ pt1 の座標をセットします。
- X 方向の尺度を入力するか対角コーナーを指定 または [コーナー (C)/XYZ(XYZ)] <1>:
  → <"">（空打ち）で既定値を受け入れます。
- Y 方向の尺度を入力 <X 方向の尺度を使用 >:　→ <"">（空打ち）で既定値を受け入れます。
- 回転角度を指定 <0>:　→ <"">（空打ち）で既定値を受け入れます。

番号	説　明
①	図面内にあるブロックを収集します。 tblnext 関数で 1 番目のブロックを取得します。それが存在した時に <while> 以下を実行します。 (cdr (assoc 2 blkname)) で得たブロック名を変数 <blkdata> にセットします。 append 関数で、こうして得たブロック名を追加してリストを作成します。
②	①と同様にして、画層名のリストを作成します。
③	blklst の中の選択された番号に相応するブロック名を変数 <sel_block> にセットします。 同様に、laylst の中の選択された番号に相応する画層名を変数 <sel_layer> にセットします。
④	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
⑤	現在の画層名を取得し (getvar "Clayer")、その名をタイトルのラベルに代入します。
⑥	リスト内容の表示は、[start_list][add_list][end_list] の 3 つを記述します。 mapcar 関数でリスト名を順番にリスト内に流し込んでいきます。
⑦	⑥と同様に、リスト内容の表示は、[start_list][add_list][end_list] の 3 つを記述します。 mapcar 関数でリスト名を順番にリスト内に流し込んでいきます。
⑧	[OK] ボタンが押されると、内部サブ関数 (SaveTileVal) を起動して、フラッグ (status) として <1> を指定します。 この数値はユーザーが指定できます。
⑨	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑩	status が <1> であれば、if 以下を実行します。<0> であれば何もせずに終了します。

Point!

tblnext 関数は、繰り返えされるたびに指定したリスト（テーブル）内の次の項目を取得します。引数の値が真（T）であれば、リスト（テーブル）の最初に戻り、次の最初の項目が取得されます。

- リスト（テーブル）に使用できる文字列は、"LAYER"、"LTYPE"、"VIEW"、"STYLE"、"BLOCK"、"UCS"、"APPID"、"DIMSTYLE"、"VPORT" があります。
- これによって取得できる値（返り値）は、DXF 形式のコードとドット・ペアのリストです。

❷ キーボードから入力する場合は、以下のように入力します。

①コマンド : (setq layname (tblnext "LAYER" T))
　返り値 → ((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "Continuous"))

この返り値の意味は次の通りです。

(0 . "LAYER")　→　シンボルタイプ
(2 . "0")　→　画層名
(70 . 0)　→　標準フラグ（<1> フリーズ解除、<2> フリーズ状態、<3> ロック状態）
(62 . 7)　→　色番号（<7> は白または黒）
(6 . "Continuous")　→　線種名（<"Continuous"> は実線）

続けて、キーボードから以下のように入力します。

②コマンド : (setq laydata (cdr (assoc 2 layname)))
　返り値 → "0" が取得できます。

続けて、キーボードから以下のように入力してリストを作成します。

③コマンド : (setq laylst (append laylst (list laydata)))
　返り値 → ("0") が返ります。

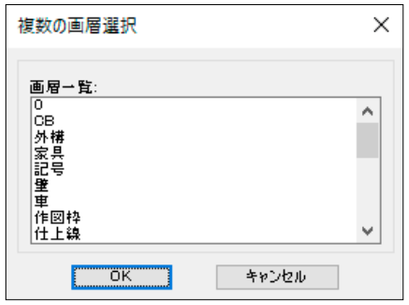
これを繰り返して、画層名のリストを作っていきます。("0" "CB" " 外構 " " 家具 "・・)

第 14 節　外部の画層ファイルを読み込み、必要な画層だけ取り込む

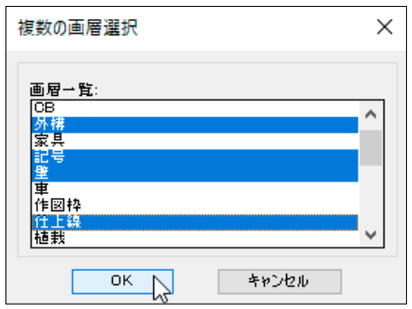
使用する LISP	S_List3.lsp	使用する主な関数	open、read_line、list、append
内部サブ関数	なし	外部サブ関数	(S_List3_1)
使用する DCL	S_List3.dcl	使用するタイル	list_box
外部テキスト	out_layer.txt	スライドファイル	なし
このプログラムでは、図面の中に画層名を記述したテキストファイルから必要な画層だけを取り入れることができます。何種類も作成しておけば、画層名の変更や追加が楽になります。			

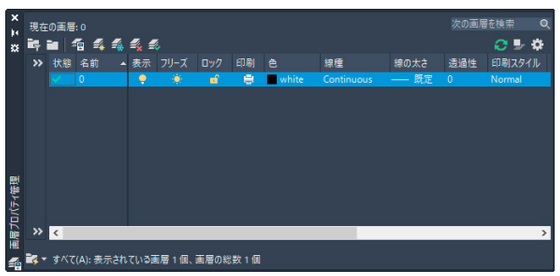
Step1 ー <S_List3.lsp> を起動すると、左下図のダイアログが表示されます。
起動した時点で、ダイアログのリスト内には画層名の一覧が表示されます。（左図）
この画層名一覧のテキストファイルは、事前に作成しておきます。（右図）

☞外部のテキストファイルから画層名を取り込む方法は、P1-190 参照

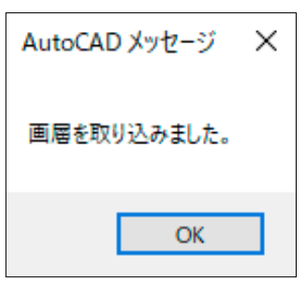


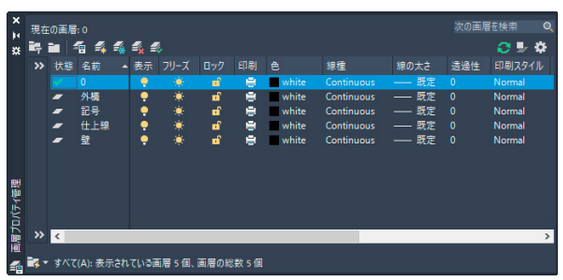
Step2 ー画層名のリストから、必要とする画層名を選択します。（左図）
右の [画層プロパティ管理] には、画層は <0> しかありません。





Step3 ー [OK] ボタンを押すと、< 画層を取り込みました。> のメッセージが表示されます。（左図）
右の [画層プロパティ管理] には、選択した画層名が追加されています。

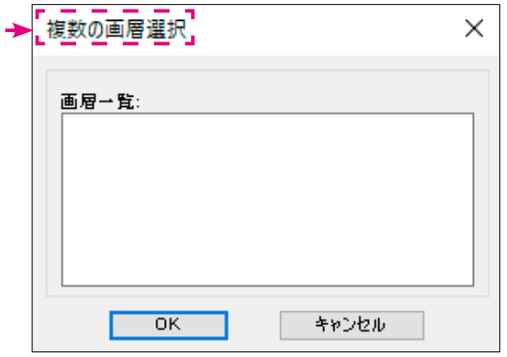


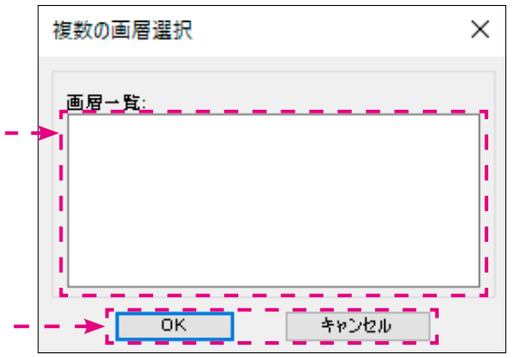


使用するダイアログ	S_List3.dcl

```
S_List3 : dialog
{
 label = " 複数の画層選択 ";
  : column    /* タイルを縦並びに配置スタート */
  {
   : boxed_column   /* タイルを縦並びに配置スタート */
   {
    : list_box
    {
     key = "layer_sel";
     label = " 画層一覧 :";
     multiple_select = true;
     width = 40;
    }
   }    /* タイルを縦並びに配置終了 */
  }     /* タイルを縦並びに配置終了 */
/* OK or CANCEL */
  ok_cancel;
}//end
```

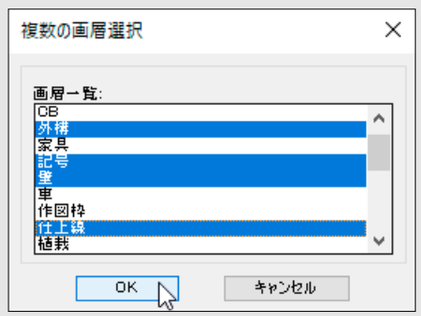
Point!





Point!

multiple_select = true;



[multiple_select=true;] を設定すると、複数の項目を取得できます。一度に複数の項目を取得するケースとしては、外部ファイルから画層を複数取り込んだり、複数のブロック図形を挿入する場合です。

取り込んだデータは、左図の画層の例では (" 外構 " " 躯体 " " 車 " " 植栽 ") ですが、リストのデータとしては (" 1 4 5 8") となっています。
これから、1 つずつ取り出してリスト番号に対応する画層名のリストに再構築します。
結果として、(" 外構 " " 躯体 " " 車 " " 植栽 ") のリストになりますから、それから 1 つずつ取り出して画層を追加していきます。

使用する LISP	S_List3.lsp

```
(defun C:S_List3( / in_layer layer_name lst dcl_id i add_layer)
;<< ファイルを読み込みモードで開く >>
❶ (setq  in_layer  (open  "out_layer.txt"  "r"))   ; テキストファイルを読み込む
  (setq  layer_name (read-line  in_layer))         ;1 行目を読み込む
  (setq  lst (LIST layer_name))                    ;1 行目をリストにする
  (while (/= layer_name nil)                       ; 画層名が無くなるまで続ける
    (setq  layer_name (read-line  in_layer))       ;1 行ずつ読み込む
    (setq  lst (append lst (LIST layer_name)))     ; 順番にリストに追加していく
  )
  (close  in_layer)                                ; テキスト ファイルを閉じる
```

→ テキストファイルから画層名を読み込み、リストを作成します。

```
;<< ダイアログを使用する準備 >>
❷ (setq dcl_id (load_dialog "S_List3"))   ;S_List3.dcl をメモリーに読み込む
  (new_dialog "S_List3" dcl_id)           ;S_List3 ダイアログを初期化する
```

→ ダイアログの読み込みと表示

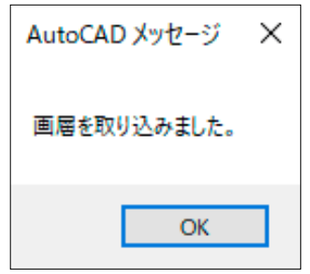
```
;<< リスト内に画層名を流し込む >>
  (start_list "layer_sel" 3)              ; リスト "layer_sel" を新規でスタート
;<<lst から 1 つずつ取り出してリストに追加 >>
  (setq i 0)                              ; カウントの初期値を <0> にセット
  (while (/= (nth i lst) nil)             ; リストが空になるまで繰り返す
❸   (setq add_layer (nth i lst))          ; 取り出した画層名をリストに追加
    (add_list add_layer)                  ; "layer_sel" 内に <lst> の内容を流し込む
    (setq i (+ i 1))                      ; カウントを <1> プラスして、繰り返す
  )
  (end_list)                              ; "layer_sel" へのリスト追加を終了
;<<"layer_sel" の値を受け取る >>
❹ (setq retlist2(list))                   ; 空のリスト (retlist2) を予め作成する（選択した複数の画層名をリスト化するため）
❺ (action_tile "layer_sel"
     "(setq readlist (get_tile ¥"layer_sel¥"))"   ; リストから選択した画層名の
  )                                               ; リスト番号を取得
;<< ダイアログを終了する準備 >>
❻ (setq dialog-box (start_dialog))        ; ユーザー入力を受け付ける
  (unload_dialog dcl_id)                  ; ダイアログをメモリーから解放する
❼ (if(= (getvar "DIASTAT") 1) (S_List3_1))   ;[Ok] ボタンが押された時の処理（外部サブ関数を起動する）

 (princ)
);end
```

作成したリストを順番に取り出して、リストボックス内に流し込みます。

"layer_sel" が操作されたときの処理を記述します。

ダイアログの入力と終了


```
❽ (defun S_List3_1( / count item retlist2 i layname)
  (setq count 1)
Point! (while (setq item (read readlist)) ; 選択したリスト (readlist) の最初の項目を読み込む
   (setq retlist2(append retList2 (list (nth item lst)))) ; 読み込んだ項目のリスト番号に相応する画層名を
   (while                                                 ; 取得し、リスト (retList2) に追加して行く
    (and
     (/= " " (substr readlist count 1))  ; 半角の空白 (" ")< スペース > では無く
     (/= ""  (substr readlist count 1))  ; 空白 ("") でもなければ。
    )
    (setq count (1+ count)) ; カウントを 1 つ増やして、次の文字を検査する（空白にぶつかるまで）
   )
   (setq readlist (substr readlist count)) ; 空白が有れば、それ以降の文字列だけを readlist に入れ直して
  )                                        ; 上から 3 行目に戻り、nil になるまで繰り返す
;<<retlist2 は (" 外構 " " 躯体 " " 車 " " 植栽 ") のようになっているので、順番に取り出して、画層を新規作成する >>
  (setq i 0)
❾ (setq layname (nth i retlist2)) ;retlist2 の 1 番目を取り出す
Point! (while (/= layname nil) ; 最後まで (nil になるまで)、以下を繰り返す
   (command "-LAYER" "n" layname "") ; 画層を新規作成する
   (setq i (1+ i)) ; カウント ( i ) を 1 加える
   (setq layname (nth i retlist2)) ; 次の画層名を取り出す
  )
  (alert " 画層を取り込みました。")
);end
```



Point!

❽ 前ページで取得した lst の内容は、以下の通りです。
("CB" " 外構 " " 家具 " " 記号 " " 躯体 " " 車 " " 作図枠 " " 仕上線 " " 植栽 " " 設備 " " その他・外形 " " タイトル " " 建具 " " 中心線 " " ハッチング " " 補助線 " " 目地 " " 文字・寸法 " nil)

(setq item (read readlist))
→ item には readlist<"1 4 5 8"> の最初の <1> が入ります。(" 外構 " " 躯体 " " 車 " " 植栽 " を選択した場合)
（read 関数は、数字の文字を自動的に整数に変換しますから、item には整数値が入ります。）

(append retList2 (list (nth item lst))
→ retList2 には lst の上から 2 番目の " 外構 " が入ります。これをリスト (" 外構 ") にします。

(and (/= " " (substr readlist count 1))(/= "" (substr readlist count 1)))
→ 1 文字ずつ取得し、その文字が半角の空白 <" ">、且つ空白 <""> で無ければ、カウントを 1 つ増やして次を検索。
list_tile の値は先頭にスペース（半角の空白 <" ">）を含んでいることがあるので、それをチェックしています。

(setq readlist (substr readlist count))
→空白にぶつかれば、それ以降を再度 readlist にセットして、最初に戻ります。
2 回目が終わると、retList2 には (" 外構 " " 躯体 ") となっています。

最後の nil になるまで繰り返された時の retList2 は、(" 外構 " " 躯体 " " 車 " " 植栽 ") となります。

Point!

❾ 前ページで取得した retlist2 の内容は、以下の通りです。
(" 外構 " " 躯体 " " 車 " " 植栽 ")

(setq layname (nth i retlist2))
→ layname には最初は <" 外構 "> が入ります。
(while (/= layname nil)
→ layname があるまで（nil が返されるまで）続けます。
(command "-LAYER" "n" layname "")
→画層を新規作成 (new) します。最後は空打ち <""> が必要です。
(setq i (1+ i))
→カウントを 1 プラスして、次の項目を取得します。
(setq layname (nth i retlist2))
→ 2 番目の項目は、" 躯体 " です。最初に戻って、nil になるまで繰り返します。

番号	説　明
①	画層名のテキストファイルを 1 行ずつ順番に読み込んでいきます。(layer_name) 1 行目を読み込んでリストにします。(lst) 最後の行まで順番に読み込み、その都度リストに追加します。
②	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
③	①で取り込んだ画層名を 1 つずつリスト内に追加していきます。 リスト内容の表示は、[start_list][add_list][end_list] の 3 つを記述します。 mapcar 関数でリスト名を順番にリスト内に流し込みます。
④	外部関数で使用するリスト <retlist2> を作成しておきます。最初は空です。
⑤	"layer_sel" の中の選択された項目の番号のリストを変数 <readlist> にセットします。
⑥	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑦	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_List3_1) を起動し、ダイアログを終了します。
⑧	multiple_select で選択した項目は、1 つの文字列になっています。 (例）"(1 4 5 8)" → 画層名に変換して (" 外構 " " 躯体 " " 車 " " 植栽 ") に再リストします。 これを行うには、read 関数と append 関数を使い、上記のリストを作成します。
⑨	選択した画層名を 1 つずつ取り出して、新規画層を作成していきます。

第 4 章　ダイアログをコントロールする

ダイアログから他のダイアログを起動したり、数値や文字列のやりとりが可能です。

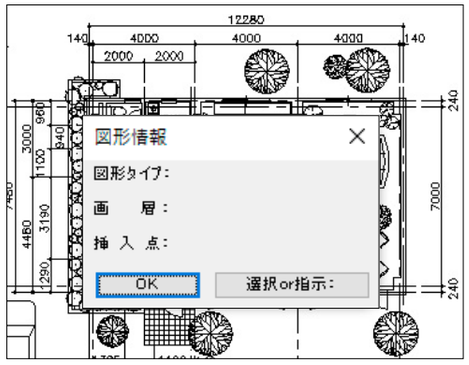
この章では、複数のダイアログをコントロールする手法を解説します。

第１節　ダイアログを一時的に隠す（図形編）

使用する LISP	S_Pick1.lsp	使用する主な関数	while、if、progn、cond、entsel
内部サブ関数	なし	外部サブ関数	なし
使用する DCL	S_Pick1.dcl	使用するタイル	text_tile
外部テキスト	なし	スライドファイル	なし
このプログラムでは、オブジェクト情報を続けて取得することができます。 一時的にダイアログは消えますが、オブジェクトを選択すると再度表示されます。			

Step1 － <S_Pick1.lsp> を起動すると、下図のダイアログが表示されます。
ダイアログの作成時には、文字は表示されていませんが、（右図）
プログラムを起動すると、初期値として３行の文字列がセットされます。（左図）

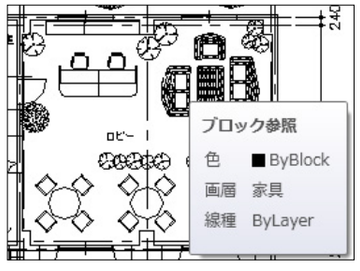
起動時の画面



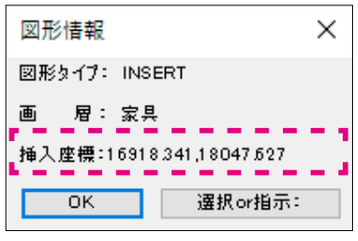
ダイアログの作成画面

Step2 － [選択 or 指示：] ボタンを押すと、ダイアログは画面から一時的に消えます。
オブジェクトを選択すると再表示され、選択したオブジェクトの情報を表示します。
3 番目の [挿入点：] の文字は、選択したオブジェクトによって変更されます。
ブロックの場合は [挿入座標]、線分の場合は [線分始点]、円の場合は [円の中心]、
その他の場合は、[挿入点] の文字に変わります。

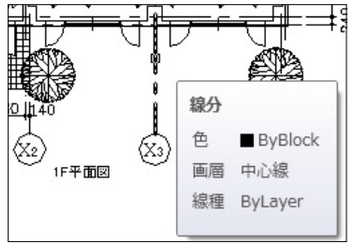
ブロックを選択



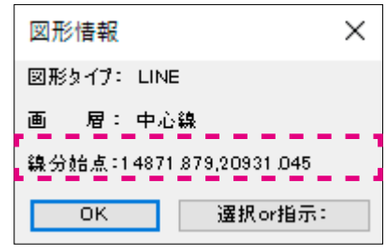
[挿入点] の名称が [挿入座標] に変更されます。



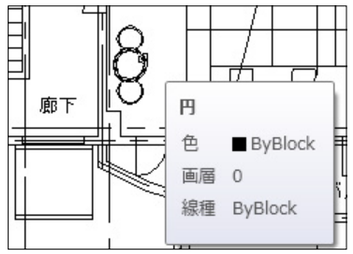
線分を選択



[挿入点] の名称が [線分始点] に変更されます。

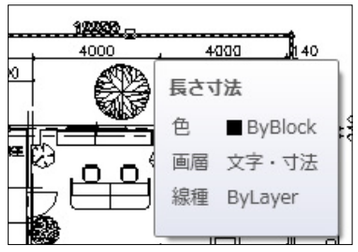


円を選択



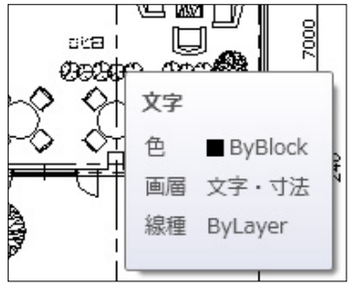
[挿入点] の名称が [円の中心] に変更されます。

寸法を選択

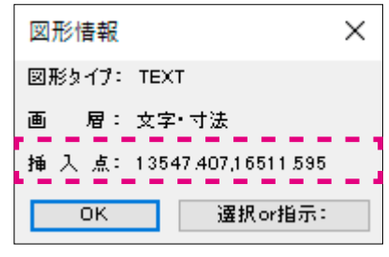


[挿入点] の名称は [挿入点] のままです。

文字を選択

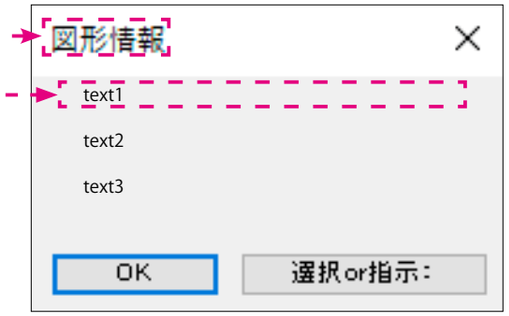


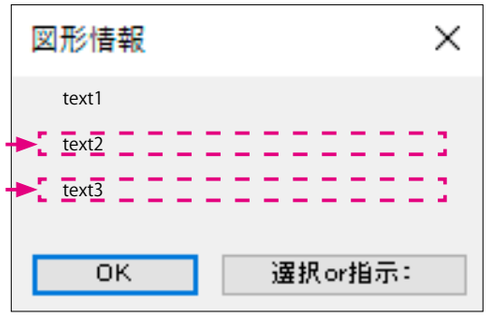
[挿入点] の名称は [挿入点] のままです。



使用するダイアログ　S_Pick1.dcl

```
S_Pick1 : dialog {
 key = "xyz";
 label = " 図形情報 ";
 : column  /* タイルを縦並びに配置スタート */
 {
  : text
  {
   label = "";
   key = "text1";
   width = 30;
  }
 spacer;
  : text
  {
   label = "";
   key = "text2";
   width = 30;
  }
 spacer;
  : text
  {
   label = "";
   key = "text3";
   width = 30;
  }
 spacer;
 }    /* タイルを縦並びに配置終了 */
 :row     /* タイルを横並びに配置スタート */
 {
  ok_only;
 :retirement_button
 {
  label = " 選択 or 指示：";
  key = "hide";
 }    /* タイルを横並びに配置終了 */
 }
}//end
```





Point!

使用する LISP　S_Pick1.lsp

```
(defun c:S_Pick1 ( / dcl_id pic_next)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Pick1.dcl"))  ;S_Pick1.dcl をメモリーに読み込む
❷ (setq pic_next 2)
  (setq entType "")
  (setq entLayer "")
  (setq entlns "")
;<< 図形をピックする度にダイアログを表示する >>
Point!
❸ (while ( = pic_next 2)                    ; 図形が選択された時の処理
   (if (null (new_dialog "S_Pick1" dcl_id))  ; ダイアログを初期化する
   (exit)                                    ; ダイアログが見つからなければ終了
   );if
;<<entType によって、「挿入点」の文字を変更する >>
❹ (if (/= entType nil)
   (progn
     (cond
       ((= entType "LINE")(setq insertPt " 線分始点："))
       ((= entType "CIRCLE")(setq insertPt " 円の中心："))
       ((= entType "INSERT")(setq insertPt " 挿入座標："))
       ( T                  (setq insertPt " 挿 入 点：  "))
     );cond
   );progn
  );if
;<< 選択した図形の情報を表示する >>
❺ (set_tile "text1" (strcat " 図形タイプ：  " entType))
  (set_tile "text2" (strcat " 画　　層：  " entLayer))
  (set_tile "text3" (strcat insertPt entlns))
;[OK] ボタンと [Cancel] ボタンの動作を記述する (done_dialog) の status をセット
❻ (action_tile "accept" "(done_dialog 1)")  ;status を <1> にセット
  (action_tile "hide" "(done_dialog 2)")    ;status を <2> にセット
;<< ダイアログを終了する準備 >>
❼ (setq pic_next (start_dialog))  ; ユーザー入力を受け付ける（入力がある度に、ダイアログを表示）
```

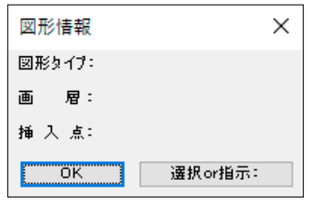
3 つの tile に表示する変数の初期値を空にセット。
status(pic_next) を <2> にセット。

status(pic_next) が <2> であれば、"S_Pick1" ダイアログを表示する。

図形が選択された時、線分であれば、" 線分始点：" の文字、円であれば、" 円の中心："、ブロックであれば、" 挿入座標："、その他は " 挿入点：" の文字を変数 <insertPt> にセットする。

"text1" ～ "text3" に選択した図形のタイプごとに文字をセット。

ダイアログの入力と終了



```
 (cond
  ((= pic_next 2)                                    ;[hide] キーが押される度に、以下の処理を行う
   (setq ent1 (entsel "¥n 図形を選択 : "))           ;[ 図形を選択 : ] の文字を表示して、図形を選択
   (setq entName (car ent1)                          ; 図形名を変数 [entName] にセット
         entList (entget entName)                    ; 図形名から図形情報を変数 [entList] にセット
         entType (cdr (assoc 0 entList))             ; 図形情報から図形タイプを変数 [entType] にセット
⑧       entLayer (cdr (assoc 8 entList))            ; 図形情報から画層名を変数 [entLayer] にセット
         entXpt (rtos (nth 1 (assoc 10 entList)) 2 3) ;X 座標値を小数点以下第 3 位まで取得
         entYpt (rtos (nth 2 (assoc 10 entList)) 2 3) ;Y 座標値を小数点以下第 3 位まで取得
         entIns (strcat entXpt "," entYpt)           ;X 座標値と Y 座標値を 1 つの文字列に連結
   );setq
  );pic_next                                         ;[Ok] ボタンを押すまで、繰り返す。
 );cond
);while
⑨ (unload_dialog dcl_id)      ; ダイアログをメモリーから解放する

(princ)
);end
```

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。
②	[pic_next] は⑥の [hide] ボタンを押したかどうかのフラッグ (status) です。<1> の時はダイアログは終了しますが、<2> の時は、ダイアログは一時的に隠れて図形を指示することが可能になります。 また、図形の結果を表示する文字列の初期値を空 <""> にセットしておきます。
③	図形が選択されるごとに、dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合は、このように (exit) で終了できます。
④	⑧で取得した図形名に従って、"text1" から "text3" に表示する文字列の準備をします。
⑤	"text1" から "text3" に取得した図形情報をセットします。 [set_tile] は [load_dialog] の後、[action_tile] の前に記述します。
⑥	[OK] ボタンが押された時は、done_dialog に <1> をセットし、[選択 or 指示 :] ボタンが押された時は、done_dialog に <2> をセットします。(<2> の時は、ダイアログは一時的に隠れます。)
⑦	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 ⑥の数値 <2> が pic_next に代入されます。
⑧	選択した図形の情報を entsel 関数で取得します。この中の [entType] によって "text3" の文字列を変更します。 その他 [画層] や [始点の座標][円の中心座標][挿入位置の座標] が取得できます。
⑨	最後に、unload_dialog を呼び出して DCL ファイルをロード解除します。

Point!

❸ 指定したダイアログが見つからなかったときの処理

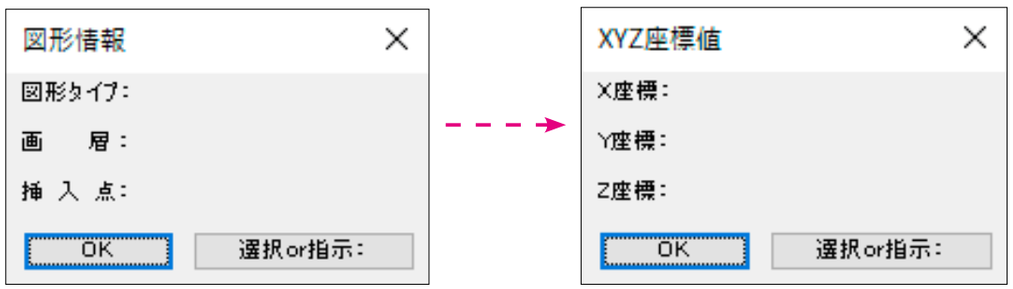
```
(while ( = pic_next 2)
  (if (null (new_dialog "S_Pick1" dcl_id)) ; 指定したダイアログが無い場合は、
  (exit)                                   ; プログラムを終了します。
  );if
```

上記のように、エラーが起きたときの回避処理 (exit) を記述しておくこともできます。

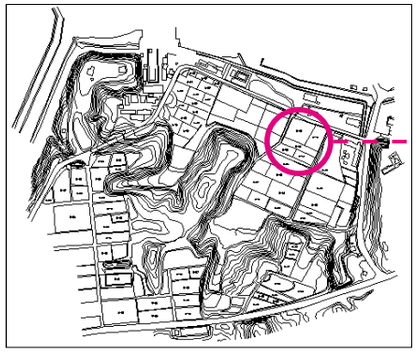
第 2 節　ダイアログを一時的に隠す（座標編）

使用する LISP	S_Pick2.lsp	使用する主な関数	while、if、progn、cond、entsel
内部サブ関数	なし	外部サブ関数	なし
使用する DCL	S_Pick1.dcl	使用するタイル	text_tile
外部テキスト	なし	スライドファイル	なし
このプログラムでは、図面内の位置情報を続けて取得することができます。一時的にダイアログは消えますが、点座標を指示すると再度表示されます。			

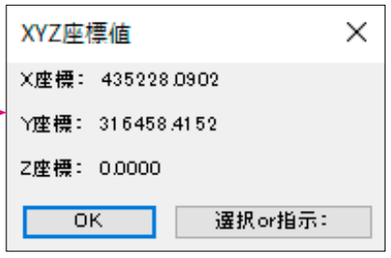
Step1 ー <S_Pick2.lsp> を起動すると、下図のダイアログが表示されます。
S_Pick1.lsp で使用したダイアログを使用します。(左図)
プログラムを起動すると、自動的にタイトルと 3 行の文字列が変更されます。(右図)

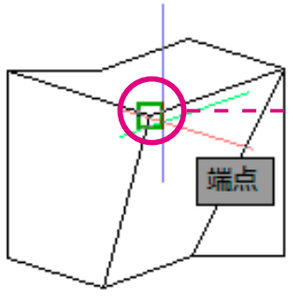


Step2 ー [選択 or 指示 :] ボタンを押すと、ダイアログは画面から一時的に消えます。
図面内の位置を指示すると再表示され、指示した位置の座標 (XYZ) を表示します。

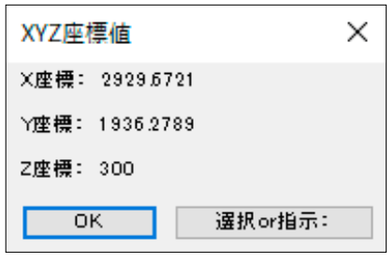


平面図では、Z 座標はたいてい <0> です。





立体図では、Z 座標には高度が入ります。



```
(defun c:S_Pick2 ( / dcl_id pic_next)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Pick1.dcl"))  ;S_Pick1.dcl をメモリーに読み込む
❷ (setq pick_next 2)
   (setq ptx "")
   (setq pty "")
   (setq ptz "")
;<< 座標をピックする度にダイアログを表示する >>
❸ (while (= pick_next 2)
    (if (null (new_dialog "S_Pick1" dcl_id))
    (exit)
    )
; ダイアログのラベルを変更する
❹ (set_tile "xyz" "XYZ 座標値 ")
;text の名称を変更する
❺ (set_tile "text1" (strcat "X 座標：    " ptx))
   (set_tile "text2" (strcat "Y 座標：    " pty))
   (set_tile "text3" (strcat "Z 座標：    " ptz))
;[OK] ボタンと [Cancel] ボタンの動作を記述する (done_dialog) の status をセット
❻ (action_tile "accept" "(done_dialog 1)")  ;status を <1> にセット
   (action_tile "hide" "(done_dialog 2)")   ;status を <2> にセット
; 入力がある度に、ダイアログを表示する
❼ (setq pic_next (start_dialog))  ; ユーザー入力を受け付ける

Point!
❽ (cond
    ((= pic_next 2)                                  ;[hide] キーが押される度に、以下の処理を行う
     (setq pt1 (getpoint "¥n 位置を指示 : "))        ; ユーザーに座標を指示するようメッセージを出す
     (setq ptx (rtos (car pt1) 2 4))                 ; 座標値の最初のリスト（X 座標）を小数点 4 桁まで取得
     (setq pty (rtos (cadr pt1) 2 4))                ; 座標値の 2 番目のリスト（Y 座標）を小数点 4 桁まで取得
     (setq ptz (rtos (caddr pt1) 2 4))               ; 座標値の 3 番目のリスト（Z 座標）を小数点 4 桁まで取得
    );what_next                                      ;[Ok] ボタンを押すまで、繰り返す。
   );cond
 );while
❾ (unload_dialog dcl_id)  ; ダイアログをメモリーから解放する

(princ)
);end
```

3 つの tile に表示する変数の初期値を空にセット。status(pic_next) を <2> にセット。

status(pic_next) が <2> であれば、"S_Pick1" ダイアログを表示する。

ダイアログのタイトルに [XYZ 座標値] を指定する。

位置が指示されたときの [X 座標値]、[Y 座標値]、[Z 座標値] を "text1" から "text3" にセットする。

ダイアログの入力と終了

番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。
②	[pic_next] は⑥の [hide] キーを押したかどうかのフラッグ (status) です。<1> の時はダイアログは終了しますが、<2> の時は、ダイアログは一時的に隠れて座標を指示することが可能になります。 また、XYZ の座標値を表示する文字列の初期値を空 <""> にセットしておきます。
③	図形が選択されるごとに、dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合は、このように (exit) で終了できます。
④	ダイアログのタイトルの文字を [XYZ 座標値] にセットします。
⑤	"text1" から "text3" に取得した座標値をセットします。 [set_tile] は [load_dialog] の後、[action_tile] の前に記述します。
⑥	[OK] ボタンが押された時は、done_dialog に <1> をセットし、[選択 or 指示：] ボタンが押された時は、done_dialog に <2> をセットします。（<2> の時は、ダイアログは一時的に隠れます。）
⑦	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。⑥の数値 <2> が pic_next に代入されます。
⑧	getpoint 関数で座標値を取得します。座標値には (X、Y、Z) の 3 つのリストから成り立っています。 X 座標を取り出し、十進法の小数点以下 4 位までを文字列に変換します。"text1" には文字列しかセットされないからです。リストの 2 番目の Y 座標と、3 番目の Z 座標も同様に文字列に変換します。
⑨	最後に、unload_dialog を呼び出して DCL ファイルをロード解除します。

Point!

❽ 指示した座標値が (252.944 338.025 158.831) の場合

(setq ptx (rtos (car pt1) 2 4))　→ "252.9437" が取得できます。
- (car pt1) → 252.944
- (rtos 252.944 2 4) → "252.9437"

(setq pty (rtos (cadr pt1) 2 4))　→ "338.0254" が取得できます。
- (cadr pt1) → 338.025
- (rtos 338.025 2 4) → "338.0254"

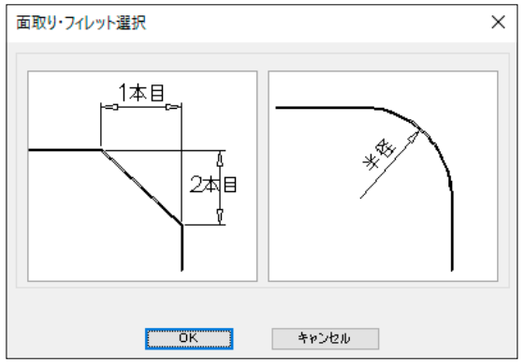
(setq ptz (rtos (caddr pt1) 2 4))　→ "158.831" が取得できます。
- (caddr pt1) → 158.831
- (rtos 158.831 2 4) → "158.831"

第 3 節　別のプログラム（ダイアログ）を起動する

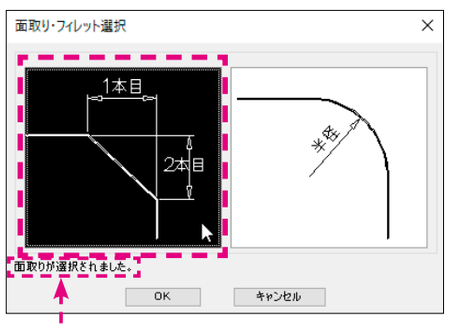
使用する LISP	S_Edit3.lsp	使用する主な関数	if、progn、cond
内部サブ関数	なし	外部サブ関数	(cha)、(fil)
使用する DCL	S_Edit3.dcl	使用するタイル	icon_image、text_tile
外部テキスト	なし	スライドファイル	S_Edit3-C、S_Edit3-F
このプログラムは、ダイアログの画像を指示して、それに関連したプログラムを起動するものです。実行するプログラムを画像で選択することで、視覚的に判りやすくヒューマン エラーのリスクを減少させることができます。			

Step1 ー <S_Edit3.lsp> を起動すると、下図のダイアログが表示されます。
ダイアログには 2 つの画像しかありません。
画像を選択することによって、その画像に関連したプログラムが起動します。

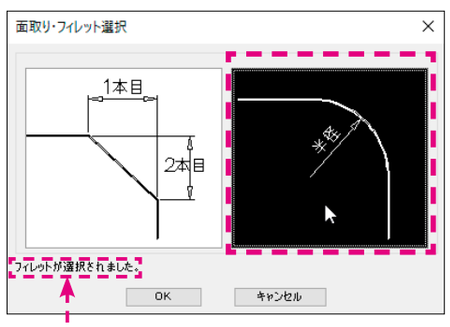
起動時の画面



Step2 ー左の画像を選択すると [面取り] のコマンドに移ります。
右の画像を選択すると [フィレット] のコマンドに移ります。
確認のために、画像の下に文字でも表示しています。



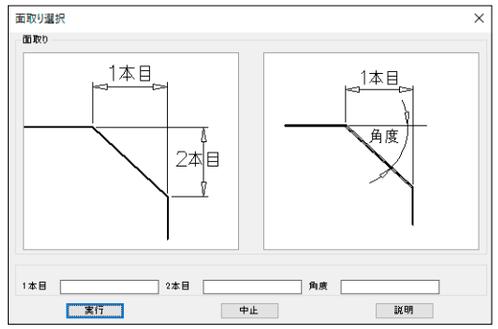
[面取りが選択されました。] と表示します。



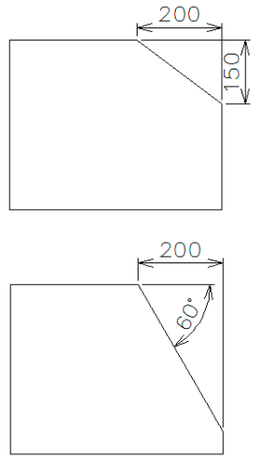
[フィレットが選択されました。] と表示します。

Step3 ー [面取り] を選択すると、第 3 章第 5 節の [S_Edit2.lsp] が起動します。
ダイアログの仕組みや AutoLISP の内容は、その節をご参考ください。
このように、1 つのダイアログを他のプログラムを起動するプラットフォームのように利用することが出来ます。

[面取り] を選択

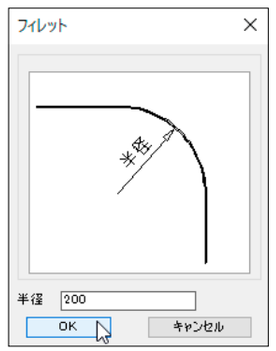


[S_Edit2.lsp] を呼び出す

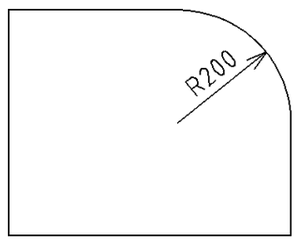


Step4 ー [フィレット] を選択すると、[S_Edit3_F.lsp] が起動します。
このプログラムは、この節の後半で解説しています。
このように画像で操作を続ければ、コマンドラインに目を落とすことも無くなりますから、時間的にも精神的にも負担が軽減されます。

[フィレット] を選択

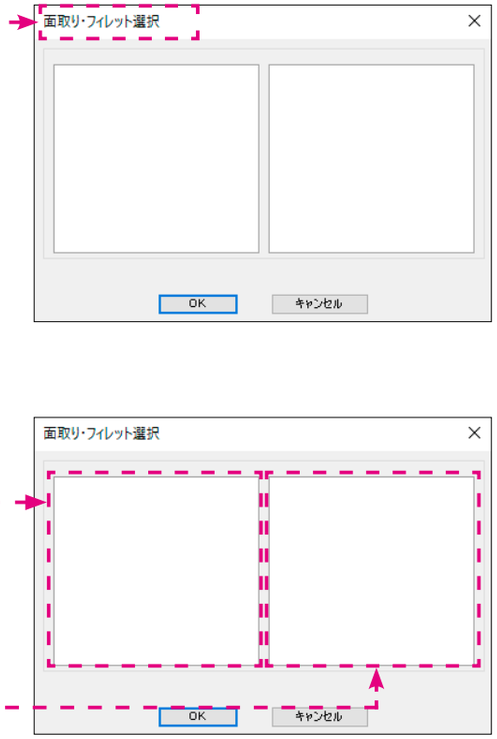


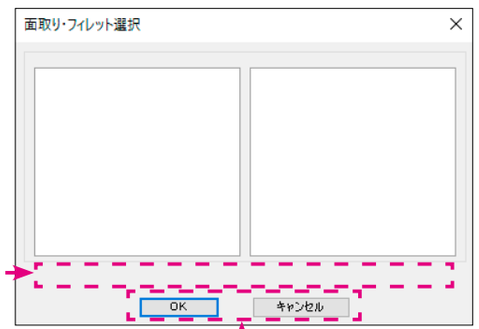
[S_Edit3_F.lsp] を呼び出す



使用するダイアログ	S_Edit3.dcl

```
S_Edit3 : dialog
{
  label = " 面取り・フィレット選択 ";

    :boxed_row  /* タイルを横並びに配置スタート */
    {
     label = "";

      :icon_image /* イメージ処理タイルスタート */
      {
        label = "";
        key = "image01";
        width = 30;
        height =15;
      }         /* イメージ処理タイル終了 */

      :icon_image /* イメージ処理タイルスタート */
      {
        label = "";
        key = "image02";
        width = 30;
        height =15;
      }         /* イメージ処理タイル終了 */

    }           /* タイルを横並びに配置終了 */

    :text
    {
      label = "";
      key = "text1";
      width = 30;
    }
     spacer;

    /* OK or CANCEL */
    ok_cancel;
}//end
```





使用する LISP	S_Edit3.lsp

```
(defun c:S_Edit3( / dialog_box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Edit3"))   ;S_Edit3.dcl をメモリーに読み込む
  (new_dialog "S_Edit3" dcl_id)           ;S_Edit3 ダイアログを初期化する
;<< イメージを表示する範囲を指定 >>
❷ (setq x1 0) (setq y1 0)                 ; スライドを表示する位置（原点は左上）
  (setq x2 (dimx_tile "image01")          ; スライドを表示する X の幅を指定する
        y2 (dimy_tile "image01")          ; スライドを表示する Y の幅を指定する
  )
;<< スライドを表示する準備 >>
❸ (start_image "image01")                 ;"image01" にスライドを読み込む
  (fill_image x1 y1 x2 y2 -2)             ; 背景色を現在の AutoCAD と同色
  (slide_image x1 y1 x2 y2 "S_Edit3-C")   ; スライド (S_Edit3-C) を表示
  (end_image)                             ; スライドの読み込みを終了
;<< スライドを表示する準備 >>
❹ (start_image "image02")                 ;"image02" にスライドを読み込む
  (fill_image x1 y1 x2 y2 -2)             ; 背景色を現在の AutoCAD と同色
  (slide_image x1 y1 x2 y2 "S_Edit3-F")   ; スライド (S_Edit3-F) を表示
  (end_image)                             ; スライドの読み込みを終了

;<< image01 処理 変数 check=1(check=1 は [ 面取り ]) >>
❺ (action_tile "image01"
    (strcat  "(setq check 1)"
             "(set_tile ¥"text1¥"  ¥" 面取りが選択されました。¥")"
    )
  )
;<< image02 処理 変数 check=2 (check=2 は [ フィレット ]) >>
❻ (action_tile "image02"
    (strcat  "(setq check 2)"
             "(set_tile ¥"text1¥"  ¥" フィレットが選択されました。¥")"
    )
  )
```

ダイアログの読み込みと表示

スライドを表示する X と Y の幅として "image01" の幅を指定しています。

スライドを読み込み、背景色を指定して、"S_Edit3-C" のスライドを指定します。

スライドを読み込み、背景色を指定して、"S_Edit3-F" のスライドを指定します。

check が <1> の時に "text1" に表示する文字を指定します。

check が <2> の時に "text1" に表示する文字を指定します。

文字列にクォーテーションを含めなければならない場合は、
クォーテーションの前に円記号を付けます (¥")
→ "(set_tile ¥"text1¥" ¥" フィレットが選択されました。¥")"

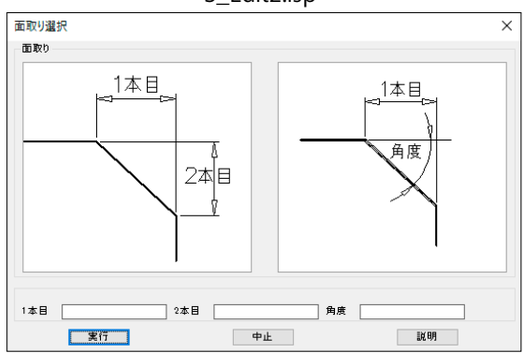
```
;<< ダイアログを終了する準備 >>
(setq dialog-box (start_dialog))   ; ユーザー入力を受け付ける
(unload_dialog dcl_id)             ; ダイアログをメモリーから解放する
(if(= (getvar "DIASTAT") 1)        ;[Ok] ボタンが押された時の処理を記述する
   (progn
    (cond
          ( (= check 1) (cha) )    ; 外部サブ関数 (cha) を起動する
          ( (= check 2) (fil) )    ; 外部サブ関数 (fil) を起動する
          (T       (cha) )         ; どちらでも無いときは、外部サブ関数 (cha) を起動する
    );cond
   );progn
  );if

(princ)
);end
;<< 外部サブ関数 >>
(defun cha()
 (load "S_Edit2")                  ;[S_Edit2.lsp] を読み込む
 (C:S_Edit2)                       ;[S_Edit2.lsp] を起動する
);end

(defun fil()
 (load "S_Edit3_F")                ;[S_Edit3_F.lsp] を読み込む
 (C:S_Edit3_F)                     ;[S_Edit3_F.lsp] を起動する
);end
```

⑦ ⑧ ⑨

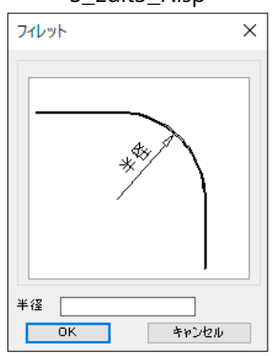
ダイアログの入力と終了

どちらを選択したかによって、起動する外部サブ関数を指定する。

S_Edit2.lsp



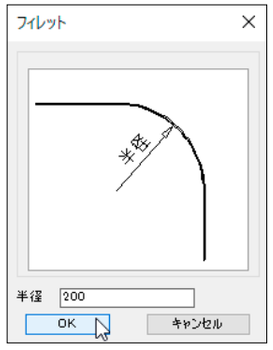
S_Edit3_F.lsp

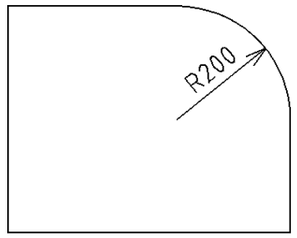


番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	スライドファイルを表示する範囲（X と Y の座標値）を指定します。この場合は、縦横の幅を dcl で設定した "image01" の数値を指定しています。
③	イメージの表示は、[start_image][slide_image][end_image] の 3 つを記述します。 [fill_image] を指定した場合は、イメージの背景色を指定できます。（<-2> はその時点の AutoCAD の背景色と同じになります。）
④	③と同様に、イメージの表示は [start_image][slide_image][end_image] の 3 つを記述します。 [fill_image] を指定した場合は、イメージの背景色を指定できます。（<-2> はその時点の AutoCAD の背景色と同じになります。）
⑤	image_tile "image01" が選択されると、プログラム分岐変数 check に <1> をセットします。 check<1> は面取りのプログラムを指定します。 また "text1" に [面取りが選択されました] の文字列をセットします。
⑥	image_tile "image02" が選択されると、プログラム分岐変数 check に <2> をセットします。 check<2> はフィレットのプログラムを指定します。 また "text2" に [フィレットが選択されました] の文字列をセットします。
⑦	start_dialog を呼び出して、ユーザーが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑧	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、cond 以下の関数 ([面取り] と [フィレット] のどちらを起動するか）を実行して、ダイアログを終了します。
⑨	分岐変数 check が <1> の時は、外部サブ関数 (cha) を起動し、check が <2> の時は、外部サブ関数 (fil) を起動します。

使用する LISP	S_Edit3_F.lsp	使用する主な関数	atof、setvar、if、progn
内部サブ関数	なし	外部サブ関数	(S_Edit3_F_1)
使用する DCL	S_Edit3_F.dcl	使用するタイル	icon_image、edit_box
外部テキスト	なし	スライドファイル	S_Edit3-F
このプログラムでは、フィレットの数値入力を画像で分かりやすくしたものです。 単独でも使用出来ます。			

Step1 － <S_Edit3_F.lsp> を起動すると、左下図のダイアログが表示されます。
入力する箇所は 1 つですが、画像を見ながら仕上がり状態を予想できます。

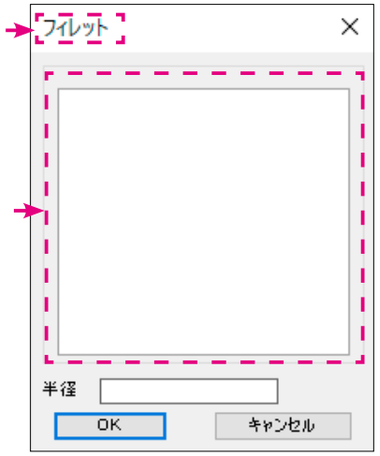


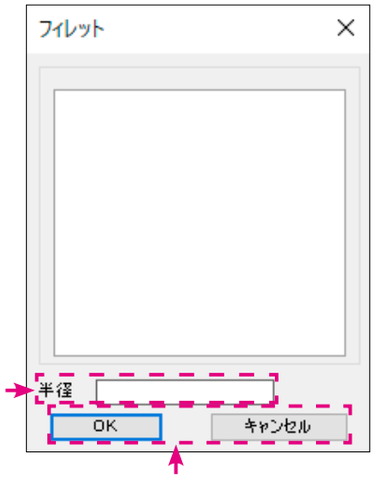


ダイアログをコントロール

使用するダイアログ	S_Edit3_F.dcl

```
S_Edit3_F : dialog
{
  label = " フィレット ";

    :boxed_row  /* タイルを横並びに配置スタート */
    {
     label = "";

      :icon_image /* イメージ処理タイルスタート */
      {
        label = "";
        key = "image01";
        width = 30;
        height =15;
      }         /* イメージ処理タイル終了 */

    }         /* タイルを横並びに配置終了 */

    :row    /* タイルを横並びに配置スタート */
    {
        :edit_box  /* 編集ボックス処理タイルスタート */
        {
          label = " 半径 ";
          key = "han";
          width = 6;
          fixed_width = true;
          value = "";
        }         /* 編集ボックス処理タイル終了 */

     spacer;
    }    /* タイルを横並びに配置終了 */

  /* OK or CANCEL */
  ok_cancel;

}//end
```





使用する LISP	S_Edit3_F.lsp

```
(defun c:S_Edit3_F( / dialog_box dcl_id)
;<< ダイアログを使用する準備 >>
❶ (setq dcl_id (load_dialog "S_Edit3_F"))   ;S_Edit3_F.dcl をメモリーに読み込む
   (new_dialog "S_Edit3_F" dcl_id)          ;S_Edit3_F ダイアログを初期化する
                                            → ダイアログの読み込みと表示

;<< イメージを表示する範囲を指定 >>
❷ (setq x1 0) (setq y1 0)                   ; スライドを表示する位置（原点は左上）
   (setq x2 (dimx_tile "image01")           ; スライドを表示する X の幅を指定する
         y2 (dimy_tile "image01")           ; スライドを表示する Y の幅を指定する
   )
                                            → スライドを表示する X と Y の幅として "image01" の幅を指定しています。
;<< スライドを表示する準備 >>
❸ (start_image "image01")                   ;"image01" にスライドを読み込む
   (fill_image x1 y1 x2 y2 -2)              ; 背景色を現在の AutoCAD と同色
   (slide_image x1 y1 x2 y2 "S_Edit3-F")    ; スライド (S_Edit3-F) を表示
   (end_image)                              ; スライドの読み込みを終了
                                            → スライドを読み込み、背景色を指定して、"S_Edit3-F" のスライドを指定します。
Point!
;<< タイル の初期設定 >>
❹ (set_tile "han" "")                       ;edit_box "han" に初期値 <""> をセットする

Point!
;<< edit_box 処理 han の値を受け取る >>
❺ (action_tile "han" "(setq han1 (get_tile¥"han¥"))")   ;edit_box "han" の値を変数 [han1] にセットする
;<< ダイアログを終了する準備 >>
❻ (setq dialog-box (start_dialog))          ; ユーザー入力を受け付ける
   (unload_dialog dcl_id)                   ; ダイアログをメモリーから解放する
                                            → ダイアログの入力と終了
;<< [OK] ボタンが押された時の処理 >>
❼ (if(= (getvar "DIASTAT") 1) (S_Edit3_F_1))   ;[OK] ボタンが押されると、システム変数 [DIASTAT] に
                                               ;<1> がセットされます。
(princ)
);end
```

Point!

❹ タイルに初期値をセットする場合

edit_box の初期値を <""> にセットするには、(set_tile "han" "") としますが
もし、具体的な数値を初期値にするときは以下のように記述します。

```
(if  (or
       ( = han1 nil)
       ( = han1 "")
       ( = han1 " "))
     (setq  han1  "200")
)
(set_tile  "han"  han1)
```


ダイアログをコントロール

```
(defun S_Edit3_F_1()
 (setq han1 (atof han1))  ;edit_box の [han1] は文字列なので、計算処理を行うために実数に変換する
 (setvar "FILLETRAD" han1)  ; フィレットの半径を指定するシステム変数 [FILLETRAD] に han1 の値をセット
 (if (/= han1 nil)  ;[han1] に値があるときの処理を記述
  (progn
   (prompt "¥n1 本目の線分と 2 本目の線分を選択：")  ; メッセージを表示する
    (command "FILLET")  ;AutoCAD の "fillet" コマンドを実行する
  );progn
 );if
 (setq han1 nil)  ; グローバル変数 [han1] の値を nil にセット
);end
```

Point!

❺

タイルの値を取得せずに [OK] ボタンを押したとき

(action_tile "han" "(setq han1 (get_tile¥"han¥"))")

→ 変数 [han1] に値が入るのは、edit_box "han" を action したときだけです。
もし、action をせずに [OK] ボタンを押すと、プログラムは何もせずに終了するか、何かエラーメッセージを表示して終了します。しかし、その時点ではダイアログは閉じていますから、ユーザーには何が原因か判りません。

このようなトラブルを回避するために、[OK] ボタンを押した後、それぞれのタイルの数値が確実に取得できているかのチェックを行って終了することが望ましいと言えます。

S_Edit3_F.lsp の一部を以下のように変更します。

; 終了する前に、このチェック関数 (CheckVal) で検査して、問題がなければ、この関数の中で
;(done_dialog 1) を記述します。

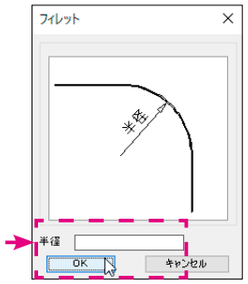
```
(defun CheckVal()
  (if (/= han1 nil)(done_dialog 1)                ; 変数 [han1] に値が有れば、ダイアログを終了
     (progn                                       ; 値が無ければ、以下を実行する
       (alert "¥n 半径が入力されていません。") ;alert 関数を起動して原因を表示する
       (mode_tile  "han"  2)                      ; カーソルのフォーカスを "han" にセットする
     );progn
  );if
);end
```

次ページで図解

```
;[OK] ボタンを押したときは、チェック関数を起動する。（ダイアログはチェック関数の中で閉じる。）
    (action_tile "accept"  "(CheckVal)") ; チェック関数 (CheckVal) を実行する。終了しない。
    (action_tile "cancel"  "(done_dialog 0)")  ; 何も処理しないで閉じる
;<< ダイアログを終了する準備 >>
    (setq chkdia (start_dialog))  ; ユーザー入力を受け付ける
    (unload_dialog dcl_id) ; ダイアログをメモリーから解放する
;<<[OK] ボタンが押された時の分岐 >>
    (if(= chkdia 1)(S_Edit3_F_1)) ; 分岐変数 [check] が <1> の時の処理を以下に記述する
```


①半径を入力せずに [OK] ボタンを押します。

(action_tile "accept" "(CheckVal)")

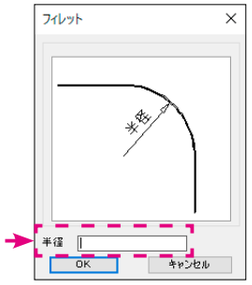


② [半径が入力されていません。] のメッセージが表示されます。

(alert "¥n 半径が入力されていません。")

③ [OK] ボタンを押すと元のダイアログに戻りますが、edit_box "han" にカーソルがフォーカスされ、入力状態になっています。

(mode_tile "han" 2)

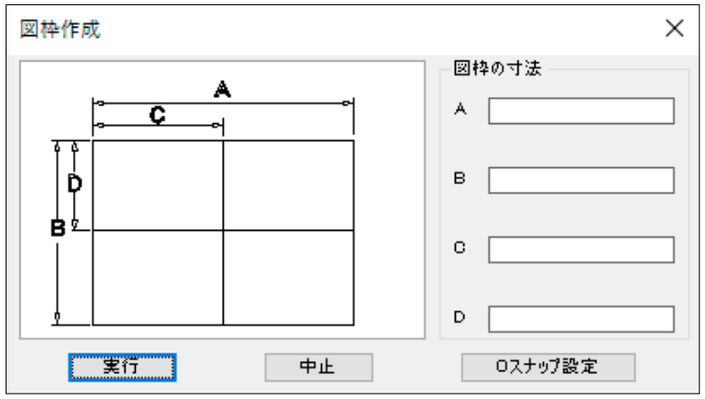


番号	説　明
①	dcl ファイルを読み込みます。dcl ファイルには複数のダイアログを含めることができます。 dcl ファイルの中のダイアログを初期化します。指定したダイアログが無い場合、(exit) の処理も記述できます。
②	スライドファイルを表示する範囲（X と Y の座標値）を指定します。縦横の幅を dcl で設定した "image01" の数値を指定しています。
③	イメージの表示は、[start_image][slide_image][end_image] の 3 つを記述します。 [fill_image] を指定した場合は、イメージの背景色を指定できます。（<-2> はその時点の AutoCAD の背景色と同じになります。）
④	フィレットの半径を指定する "han1" の初期値に <"">(nil) をセットします。 [set_tile] は [load_dialog] の後、[action_tile] の前に記述します。
⑤	edit_box "han" の内容を変数 "han1" に代入します。（文字列なので、使用する時は実数に変換します。）
⑥	start_dialog を呼び出して、ユーザが入力できるよう、コントロールをダイアログ ボックスに切り替えます。 終了ボタンを選択したときに done_dialog を呼び出すアクションが起動され、start_dialog は値を返します。 ここで、unload_dialog を呼び出して DCL ファイルはロード解除されます。
⑦	[OK] ボタン <accept> が押された時は、システム変数 [DIASTAT] に <1> がセットされます。 もし [DIASTAT] が <1> であれば、外部サブ関数 (S_Edit3_F_1) を起動し、ダイアログを終了します。

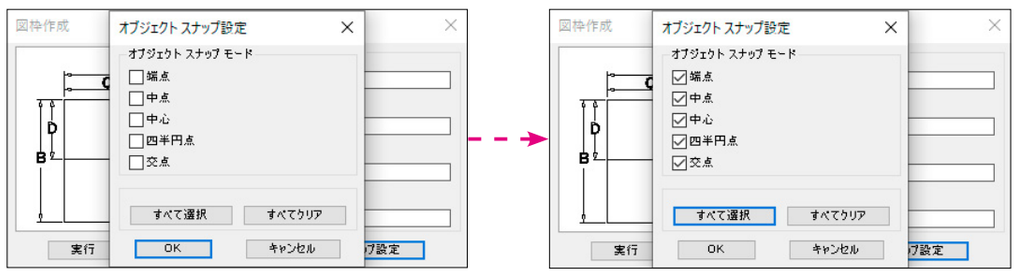
第 4 節　[Help] にコマンドを割り当てる

使用する LISP	S_Edit4.lsp	使用する主な関数	getvar、atof、polar、ssadd、entget
内部サブ関数	なし	外部関数	S_Toggle1.lsp
使用する DCL	S_Edit4.dcl	使用するタイル	icon_image、edit_box
外部テキスト	なし	スライドファイル	S_Edit1
このプログラムでは、AutoCAD の [help] ボタンの位置に、ユーザーが作成した関数を割り当てています。[help] ボタンは、このようにカスタマイズ可能です。			

Step1 ー <S_Edit4.lsp> を起動すると、下図のダイアログが表示されます。
このプログラムは、第 3 章 4 節の [S_Edit1.lsp] を改変したものです
ダイアログの右下に [O スナップ設定] のボタンを配置し、このボタンから第 3 章 9 節の [S_Toggle1.lsp] を起動させることができます。



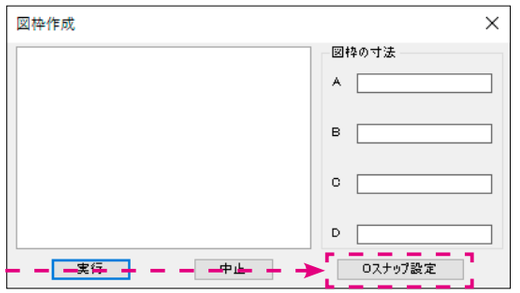
Step2 ー [O スナップ設定] ボタンを押すと、現在のダイアログの上に、[S_Toggle1.lsp] で使用した [O スナップ設定] のダイアログが表示されます。（AutoCAD の < 割り込み > と同じです。）
そのあと、このダイアログを閉じて元のダイアログで作業を継続します。



使用するダイアログ	S_Edit4.dcl

第 3 章 4 節の [S_Edit1.dcl] を利用し、OK ボタン以下を改変しました。
[OK_Cancel] に [help] を追加しています。

```
/* OK or CANCEL */
: row
{
   spacer;
  : button
  {
      label = " 実行 ";
      is_default = true;
      key = "accept";
      width = 8;
      fixed_width = true;
  }
   spacer;
  : button
  {
      label = " 中止 ";
      is_cancel = true;
      key = "Cancel";
      width = 4;
      fixed_width = true;
  }
   spacer;
  : button
  {
      label = "O スナップ設定 ";
      key = "help";
      width = 4;
      fixed_width = true;
  }
   spacer;
 }
}//end
```



[O スナップ設定] ボタンを追加

使用する LISP	S_Edit4.lsp

第 3 章 4 節の [S_Edit1.lsp] を利用し、[help] ボタンの機能を追加しました。（追加箇所のみ記述）
[help] にユーザー関数 (S_Toggle1.lsp) を割り当てています。

```
;<< edit_box 処理 tate2 に値を受け取る >>
 (action_tile "tate2"
     "(setq tate2 (get_tile ¥"tate2¥"))"
 )
;<< O スナップ設定 (S_Toggle1.lsp) を追加する >>
 (action_tile "help"
    (strcat
       "(load ¥"S_Toggle1¥")" ;[S_Toggle1.lsp] を読み込む
       "(C:S_Toggle1)"        ;[S_Toggle1.lsp] を実行する
    )
 )
;<< ダイアログを終了する準備 >>
 (setq dialog-box (start_dialog))  ; ユーザー入力を受け付ける
 (unload_dialog dcl_id)
;<< [OK] ボタンが押された時の処理 >>
 (if(= (getvar "DIASTAT") 1) (S_Edit1_1))
```

Point!

❶

"help" が押された時は、独立関数 (S_Toggle1.lsp) を読み込み実行する。

ダイアログの入力と終了

番号	説　明
①	[help] ボタンを押した時の動作を記述します。 独立関数 (S_Toggle1.lsp) を使用するときは、まずメモリーに読み込みます。 次に <C:S_Toggle1> と入力します。

Point!

ユーザー関数 (C:***.lsp) を読み込む場合

AutoCAD の組み込み関数 (Line や Circle) の場合は、コマンドラインに [LINE] または [CIRCLE] と入力するとコマンドは使えます。
しかし、ユーザー独自の関数では、メモリーに読み込む（ロードする）ことが必要です。

①コマンドラインに次の様に入力します。
　→ (load "S_Toggle1")
②読み込んだ後、コマンドを実行します。
　→ S_Toggle1

索引　&　付録

■ ■ ■ 索引 (Index)

- 本書で作成した AutoLISP & Dialog
- 本書で使用した AutoLISP 関数 & システム変数
- 本書で使用した一般用語

解説プログラム一覧と取得方法

本書に掲載した AutoLISP と DCL は下記の Web から取得できます。また、プログラムの修正文等も掲載しています。

https://www.ellipse.ne.jp/autolisp.html

第 1 部で作成した AutoLISP プログラム一覧表

章	LISP	ページ
1 章	box1.lsp	1-39
	fillet0.lsp	1-41
2 章	box2.lsp	1-49
	tbl_lay.lsp	1-60
	moji_len.lsp	1-68
	dist1.lsp	1-73
	dist2.lsp	1-75
	ang.lsp	1-79
	nitoubun.lsp	1-80
	arc_len.lsp	1-86
	polar_1.lsp	1-90
	polar_2.lsp	1-92
	para1.lsp	1-94
	ang_3hen.lsp	1-96
	ucs_o.lsp	1-113
	sui_lin.lsp	1-117
	re_copy.lsp	1-127
	whcircle.lsp	1-129
	while_off.lsp	1-130
	chusin.lsp	1-134
	chgcircle.lsp	1-137

章	LISP	ページ
3 章	trim_1.lsp	1-143
	rota1.lsp	1-154
	mo_ro.lsp	1-158
	box2.lsp	1-164
	waku_1.lsp	1-168
	waku_2.lsp	1-169
	orth_on.lsp	1-171
	off_3.lsp	1-176
	en_hasi.lsp	1-182
	out_var.lsp	1-191
	out_layer.lsp	1-192
	in_var.lsp	1-194
	in_layer.lsp	1-196

章	LISP	ページ
4 章	get_lwpoly2.lsp	1-212
	get_lwpoly.lsp	1-213
	get_polyline.lsp	1-221
	chglay.lsp	1-224
	laychg.lsp	1-226
	laychg2.lsp	1-228
	txt_ro.lsp	1-230
	DimUp.lsp	1-238
	del_lay.lsp	1-242
	m_chglay.lsp	1-244
	chg_txt.lsp	1-247
	nagachk.lsp	1-250
	circleerase.lsp	1-258
	lineerase.lsp	1-260
	xdata_add.lsp	1-265
	get_xdata2.lsp	1-268
	get_xdata3.lsp	1-269

第 2 部で作成した AutoLISP & DCL プログラム一覧表

章	節	LSP	DCL	スライドファイル (sld)	外部テキスト
第 3 章	第 2 節	S_Text2.lsp	S_Text2.dcl		
	第 3 節	S_Text1.lsp	S_Text1.dcl		
	第 4 節	S_Edit1.lsp	S_Edit1.dcl	S_Edit1	
	第 5 節	S_Edit2.lsp	S_Edit2.dcl	S_Edit2-1、S_Edit2-2	
	第 6 節	S_Slider1.lsp	S_Slider1.dcl	S_Slider1-1 ～ S_Slider1-5	
	第 7 節	S_Radio1.lsp	S_Radio1.dcl		
	第 8 節	S_Radio2.lsp	S_Radio2.dcl		
	第 9 節	S_Toggle1.lsp	S_Toggle1.dcl		
	第 10 節	S_Popup2.lsp	S_Popup2.dcl		
	第 11 節	S_Toggle2.lsp	S_Toggle2.dcl		
	第 12 節	S_List1.lsp	S_List1.dcl		
	第 13 節	S_Popup1.lsp	S_Popup1.dcl		
	第 14 節	S_List3.lsp	S_List3.dcl		out_layer.txt
第 4 章	第 1 節	S_Pick1.lsp	S_Pick1.dcl		
	第 2 節	S_Pick2.lsp	S_Pick1.dcl		
	第 3 節	S_Edit3.lsp	S_Edit3.dcl	S_Edit3-C、S_Edit3-F	
		S_Edit3_F.lsp	S_Edit3_F.dcl	S_Edit3-F	
	第 4 節	S_Edit4.lsp	S_Edit4.dcl	S_Edit1	

本書で使用した AutoLISP 関数 & システム変数　アルファベット順

記号	
!	1-19
～	1-62
/	1-62、1-63
/=	1-120、1-121
+	1-62、1-63
-	1-62、1-63
=	1-120、1-121
<	1-120、1-121
<=	1-120、1-121
>	1-120、1-121
>=	1-120、1-121
*	1-62、1-63
error	1-198、1-201
1+	1-62、1-63
1-	1-62、1-63

A	
abs	1-62、1-65
acad_strlsort	1-46、1-54
acadlspasdoc	1-6
action_tile	2-35、2-44
add_list	2-62
alert	1-198、1-205
and	1-120、1-121
angbase	1-33
angdir	1-33
angle	1-72、1-76、1-77
angtof	1-100、1-103
angtos	1-100、1-106、1-107、1-109
aperture	1-29、1-173
append	1-46、1-50、1-261
apply	1-46、1-51
assoc	1-46、1-47、1-56
atan	1-62、1-64
atof	1-100、1-103
atoi	1-100、1-103
atoms-family	1-44
aunits	1-33、1-104
auprec	1-34、1-105

B	
blipmode	1-29

C	
car	1-46、1-47
cadr	1-46、1-47
cddr	1-46、1-47
cdr	1-46、1-47

C	
cdate	1-100
cecolor	1-36
celtscale	1-36
celtype	1-36
chamfera	1-35
chamferb	1-35
chamferc	1-35
chamferd	1-35
chammode	1-35
clayer	1-36
close	1-188、1-189
cmddia	1-29
cmdecho	1-29、1-140、1-146
cond	1-136
cons	1-46、1-50、1-57
cos	1-62、1-64
cvunit	1-100、1-118

D	
defun	1-12
dimx_tile	2-60、2-97
dimy_tile	2-60、2-97
distance	1-72、1-73、1-162
distof	1-100、1-103
done_dialog	2-34
dragmode	1-29
dtr	1-78、1-79

E	
elevation	1-52
end_image	2-60
end_list	2-62
entdel	1-208、1-222
entget	1-208、1-217
entlast	1-208、1-209
entmake	1-57、1-208、1-222
entmakex	1-208
entmod	1-58、1-208、1-223、1-261
entnext	1-208、1-219
entsel	1-208、1-210
entupd	1-208
eq	1-120、1-123
eqal	1-120、1-123
exit	1-198
exp	1-62
expt	1-62、1-63
extmax	1-32
extmin	1-32

F	
filedia	1-29
fill_image	2-60
filletrad	1-35
fillmode	1-30
findfile	1-188、1-189
fix	1-100、1-101、1-102
float	1-100、1-101
foreach	1-46、1-51

G	
gcd	1-62
get_tile	2-35、2-38
getangle	1-152、1-155、1-157
getcorner	1-152、1-155
getdist	1-152、1-156、1-162
getint	1-152、1-153
getkword	1-152、1-167
getorirnt	1-152、1-155
getpoint	1-52、1-152、1-156、1-161
getreal	1-152、1-153
getstring	1-152、1-153
getvar	1-28、1-38
graphscr	1-140、1-144

H	
highlight	1-30

I	
if	1-132
initget	1-152、1-166
inters	1-72、1-98
itoa	1-100、1-106

L	
last	1-46、1-47
lastpoint	1-36、1-161
length	1-46、1-59
limmax	1-32
limmin	1-32
list	1-19、1-50、1-52
listp	1-46、1-59
load_dailog	2-34
log	1-62
logand	1-62
logior	1-62
lsh	1-62
ltscale	1-36
lunits	1-34、1-104
luprec	1-34、1-104
lwpolyline	1-211、1-212、1-213

M	
mapcar	1-46、1-50
max	1-62、1-65
member	1-46、1-54
min	1-62、1-65
minusp	1-62、1-65
mirrtext	1-30
mode_tile	2-35、2-40

N	
nentsel	1-208、1-211
new_dialog	2-34
nil	1-42、1-43
not	1-120、1-121
nth	1-46、1-47
null	1-120、1-121

O	
open	1-188、1-189
or	1-120、1-121
orthomode	1-170、1-171
osmode	1-37、1-174
osnap	1-72、1-170

P	
pause	1-42
pdmode	1-30
pdsize	1-30
pi	1-42、1-43
pickbox	1-30
plinewid	1-31
polar	1-72、1-76、1-77
prin1	1-140、1-141
princ	1-140、1-141
print	1-140、1-141
progn	1-133
prompt	1-140
psltscale	1-37

Q	
qtext	1-144
qtextmode	1-31
quit	1-198
quote	1-53

R	
read	1-66、1-71
read_char	1-188、1-189
read_line	1-188、1-189
redraw	1-140、1-144、1-145
regapp	1-261、1-263

R	
rem	1-62、1-65
repeat	1-126
reverse	1-46、1-54
rtd	1-78、1-79
rtos	1-100、1-106、1-107、1-108

S	
set_tile	2-35、2-36
setq	1-19
setvar	1-28、1-40
sin	1-62、1-64
slide_image	2-60
snapang	1-84
snapbase	1-31
snapunit	1-31
sqrt	1-62、1-63
ssadd	1-235、1-239
ssdel	1-235、1-239
ssget	1-235、1-236
sslength	1-235、1-240
ssmemb	1-235、1-240
ssname	1-235、1-240
start_dialog	2-34
start_image	2-60
start_list	2-62
strcase	1-66、1-67
strcat	1-66、1-67
strlen	1-66、1-67
subrs	1-25
subst	1-46、1-54、1-58
substr	1-66、1-67

T	
T	1-42、1-43
tblnext	1-46、1-60
tblsearch	1-46、1-59
terpri	1-140、1-141
textbox	1-72
textpage	1-140
textscr	1-140、1-144
trans	1-100、1-110、1-112、1-114
type	1-25

U	
U	1-206
UCS	1-110、1-111
unload_dialog	2-34

W	
wcmatch	1-66、1-71
while	1-129
write-char	1-188、1-189
write-line	1-188、1-189

本書で使用した一般用語

アイウエオ順

著者プロフィール

中森　隆道（なかもり　たかみち）

株式会社エリプス

・CAD&CAE プログラム開発と職業訓練校運営

・AutoCAD アドオンソフト開発

著書

・CAD 操作マニュアル分野

[Autodesk Mechanical Desktop の使い方] ソフトバンククリエイティブ株式会社

[IntelliCAD 2002 公式ガイドブック] 株式会社　鳥影社

[CAD 総合講座 AutoCAD2000i コマンド編＆練習編] ソフトバンククリエイティブ株式会社

[基礎から学ぶ AutoCAD2002 コマンド編＆実践編] ソフトバンククリエイティブ株式会社

他　社会人スクール用テキスト等　多数

・CAD 資格試験対策分野

[CAD 利用技術者試験 1 級ハンドブック 2000 年] ソフトバンククリエイティブ株式会社

[CAD 利用技術者試験 1 級ハンドブック 2001 年] ソフトバンククリエイティブ株式会社

[AutodeskMaster2DDedign 公式ガイドブック（共著）ソフトバンククリエイティブ株式会社

[AutodeskMaster3DDedign 公式ガイドブック（共著）ソフトバンククリエイティブ株式会社

以下は 2002 年から 2005 年まで毎年

[CAD 利用技術者試験 2 級徹底理解] ソフトバンククリエイティブ株式会社

[CAD 利用技術者試験 1 級徹底理解] ソフトバンククリエイティブ株式会社

他　多数

・CAD プログラム開発分野

[AutoCAD LT2000i 徹底活用] ソフトバンククリエイティブ株式会社

[AutoCAD LT2002 徹底活用] ソフトバンククリエイティブ株式会社

[AutoCAD LT データ 徹底活用術] ソフトバンククリエイティブ株式会社

[AutoCAD & LT カスタマイズブック] ソフトバンククリエイティブ株式会社

本書掲載プログラムの掲載 Web ページ　https://www.ellipse.ne.jp/autolisp.html

本書の質問メールアドレス　autolisp@ellipse.ne.jp